1991

On implementing the arithmetic Fourier transform (AFT).

Jamal. Benzreba
University of Windsor

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

ON IMPLEMENTING THE ARITHMETIC FOURIER TRANSFORM (AFT)

by

Jamal Benzreba

A Thesis
Submitted to the Faculty of Graduate Studies through the
Department of Electrical Engineering in Partial Fulfillment
of the Requirements for the Degree of
Master of Applied Science at the
University of Windsor

Windsor, Ontario
September, 1991

ISBN 0-315-69878-0

ABSTRACT

This work investigates the VLSI implementation of the Arithmetic Fourier Transform (AFT) algorithm. The study proposes several different new architectures suitable for VLSI implementation of the AFT in Digital Signal Processing (DSP) applications. As a verification tool, Extend™ is used to simulate the hardware realization of these proposed architectures.

The motivation for this work is the recent rediscovery of the algorithm, by Tufts and Sadasiv [1], which has the possibility of eliminating many of the multiplications usually associated with computing discrete Fourier coefficients. The AFT implementations are based on nearest neighbour and linear interpolation procedures.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to ***Dr. G. A. Jullien, Dr. W. C. Miller,*** *and* ***Dr. N. M. Wigley*** *for their tremendous support and guidance throughout the progress of this thesis. In addition, I would also like to thank* ***Bruce Erickson*** *for many favours.*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Digital signal processing (DSP) is a collection of computer algorithms and systems, and is concerned with the representation of signals by sequences of numbers and the processing of these sequences. The aim of such processing may be to estimate certain parameters of a signal or to transform a signal into a form which is more desirable. The importance of signal processing is evident in such diverse fields as speech, communications, seismology, sonar, radar, and many others.

One of the important tools of digital signal processing is the Discrete Fourier Transform (DFT). The DFT has played an important role in the analysis, design, and the implementation of digital signal processing algorithms and systems. There is an extensive literature about computing the discrete Fourier transform and the hardware implementations of the different algorithms. One particular type of algorithm, the fast Fourier transform algorithm (FFT) [2], has been widely accepted as the superior candidate for hardware implementation, and many variations of the basic FFT [3-10] have appeared. With the increasing emphasis on hardware based systems, which very often operate at the highest possible speeds and achieve the maximum possible

throughput, different techniques have been developed to reduce the computational complexity of the DFT. The purpose of these techniques is to simplify the arithmetic operations and, in particular, multiplication. Multiplication is extremely time consuming and much more costly than an addition. However, due to the current advanced technologies, very fast parallel multipliers are now available. Due to the relatively large area required by a multiplier in an IC realization, many proposed approaches of implementing the DFT have emphasized reducing the required number of multipliers. Further, the technology required in some applications requiring high speeds is quite advanced. Therefore, a clear-cut need exists for Fourier techniques which offer increased processing power without being too dependent on technology for high performance systems.

This thesis considers a new technique for Fourier analysis and synthesis. This new technique is the Arithmetic Fourier Transform (AFT).

1.2 HISTORICAL OVERVIEW OF THE AFT

The Fourier representation of a periodic signal is important to Fourier analysis and digital signal processing. The most efficient method of computing the Fourier transform is the Fast Fourier Transform (FFT). However, computation of the FFT is still complicated and time consuming, especially in terms of the number of needed complex multiplications.

The Arithmetic Fourier Transform seems to go back to Chebyshev [11]. However, at the beginning of this century, in 1903, a mathematician by the name Bruns [12] developed the method of computing the Fourier coefficients of a periodic signal using a number-theoretic approach. This approach is based on the Möbius inversion of series. As a mathematical theory the method has been treated quite thoroughly in Wintner's monograph [13], which discusses

the connections between, among other things, Fourier Analysis, the Arithmetic Fourier Transform, and the Prime Number Theorem.

Recently, Tufts and Sadasiv [1,14] rediscovered the same algorithm. They called this method of analysis the Arithmetic Fourier Transform (AFT). This AFT is based on the Möbius inversion of series and can be used to compute finite Fourier coefficients of symmetric periodic function. The AFT developed in [1] was extended very recently by Reed et al. [15] to compute the Fourier coefficients of any complex-valued periodic function. Practical techniques for overcoming some of the deficiencies associated with using massively oversampled data for computing the exact AFT were also presented in [15].

Up to now, there has not been any published article which investigates in detail the implementation of the AFT algorithm in the VLSI medium. Since exact computation of the AFT requires oversampling of the input signal (as we will see in chapter 2), Tufts and Sadasiv suggested an implementation based on Delta Modulation (DM) and Adaptive Delta Modulation (ADM) approaches [1,14]. The basic VLSI building block for the DM implementation [14] consists of an up-down counter, an accumulator, and timing control. The authors point out that Kouvaras [16] has shown how addition and multiplication by a scalar of delta-modulated sequences can be derived using the DM sequence for the individual signals as inputs to a simple logic network. However, they did not show, in their article, how to account for the complicated indexing of the samples required to compute the AFT, since not all of the samples are needed in the AFT computation (this is based on the supposition that DM is based on the uniform sampling of the input analog signal). The ADM is used to increase the dynamic range of the resulting transformation. Shih et al. [17] suggested a systolic implementation for the AFT algorithm which is based on Brun's alternating average; and which

utilizes a technique for the decomposition of these averages. However, it is found that the average decomposition technique can only be applied to a certain subset of the needed averages, and hence the method failed to account for the rest of the averages.

Therefore, this thesis considers, in addition to the implementation proposed by [18], new proposed architectures for implementation of the AFT algorithm into the VLSI medium.

1.3 THESIS ORGANIZATION

This thesis consists of five chapters. The first chapter serves to present the background which initiated this work and to lay out the structure of the remaining chapters. Chapter two provides an introduction to the Arithmetic Fourier Transform (AFT) [19], as it is not found in any of the usual textbooks on Fourier Analysis. Following the derivation of the exact AFT algorithm, approximate AFT computational techniques [18] to overcome the problem of requiring to massively oversample the signal are developed. It is shown that the accuracy of the approximate AFT is of a similar order of magnitude as one can obtain through the DFT. This chapter concludes with a discussion on the computational complexity of the AFT algorithm, in comparison with the familiar radix-2 FFT algorithm.

Chapter three serves to present several hardware architectures suitable for VLSI implementation of the AFT algorithm. First, an architecture proposed by Wigley and Jullien [18] is presented. Next, several new architectures based on zero order and linear interpolation procedures are proposed for implementation of the AFT.

Chapter four provides simulations of these proposed architectures using Extend™. (Extend is a simulation tool used to verify the correctness of the

proposed architectures). For each architecture, discussion on the total hardware requirements, dynamic range growth, quantization noise due to finite wordlength computation, and critical path is presented. Finally, a comparison between the proposed architectures and between these architectures and the FFT is presented.

The final chapter concludes the thesis by stating the results derived in the previous chapters. Suggestions are also made in this chapter to give direction to future research in this area.

CHAPTER 2

ON COMPUTING THE ARITHMETIC FOURIER TRANSFORM (AFT)

2.1 INTRODUCTION

In this chapter we will present both the original and approximate AFT computational techniques [18]. The problems of oversampling of the signal needed to compute the exact AFT are outlined, and practical techniques to overcome these problems are presented. A technique [15] to reduce the computational requirements of computing the imaginary component of the AFT is also presented. The AFT and delta modulation (DM) [18] is discussed. Some examples are presented to show the kind of accuracy one may expect in computing Fourier coefficients with the AFT. Finally, a comparison between the AFT and the FFT in terms of computational complexity is presented.

2.2 THE ARITHMETIC FOURIER TRANSFORM (AFT)

The Arithmetic Fourier Transform is an alternative method of computing the Fourier coefficients of a complex periodic function f(t). The method involves the computation of large numbers of Riemann sums, sums which approximate the integral of the function f(t) over one period. The Fourier coefficients are then obtained from certain linear combinations of

these sums which are formed through the use of the number-theoretic Möbius function. There are no multiplications involved, other than those multiplications which are part of the linear combinations; but since the Möbius function takes only the values 0 and ±1, these multiplications are actually merely additions or subtractions. There are divisions, but they are few in number, and the divisors are limited to a small range of positive integers. Thus it is hoped, thanks to the elimination of multiplications and the small number of integer divisions, that the method may offer a means of computing Fourier coefficients of a signal in a rapid and efficient manner.

Let f(t) be a periodic complex-valued function with period 1, so that we can write

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{2\pi kit} \tag{2.1}$$

The method requires that the term c_0 be equal to zero, so we replace f(t) with g(t), given by

$$g(t) = f(t) - c_0 = f(t) - \int_0^1 f(s)ds \tag{2.2}$$

Observe that f and g have the same Fourier coefficients except, of course, c_0. Thus there is no loss of generality in treating g(t) in lieu of f(t).

Let n be a positive integer and introduce the average of g(t) over n points:

$$S_g(n,t) = S(n,t) = \frac{1}{n} \sum_{j=0}^{n-1} g\left(t + \frac{j}{n}\right) \tag{2.3}$$

Since f, and therefore g, are periodic with period 1, the above formula gives, for each positive integer n, a function S which is periodic in t with period $\frac{1}{n}$,

i.e. $S(n, t+\frac{1}{n}) = S(n,t)$ for any t. Note also that S(n,t) is the Riemann sum of order n that approximates $\int_0^1 g(t)dt$.

We now observe that S(n,t) only depends on a certain subset of all of the Fourier coefficients, namely the subset of coefficients whose subscript is divisible by n:

$$\begin{aligned} S(n,t) &= \frac{1}{n}\sum_{j=0}^{n-1}\sum_{k=1}^{\infty}(c_k e^{2\pi ki(t+j/n)} + c_{-k}e^{-2\pi ki(t+j/n)}) \\ &= \sum_{k=1}^{\infty} c_k e^{2\pi kit}\frac{1}{n}\sum_{j=0}^{n-1} e^{2\pi kij/n} + \sum_{k=1}^{\infty} c_{-k}e^{-2\pi kit}\frac{1}{n}\sum_{j=0}^{n-1} e^{-2\pi kij/n} \\ &= \sum_{k=1}^{\infty}(c_k e^{2\pi kit} + c_{-k}e^{-2\pi kit}) \\ &\quad n \mid k \\ &= \sum_{p=1}^{\infty}(c_{np}e^{2\pi npit} + c_{-np}e^{-2\pi npit}) \end{aligned} \qquad (2.4)$$

wherein the notation $n \mid k$ means that only those terms shall contribute to the sum for which n divides k. In the derivation of this formula we have used the fact that the sum of the first n powers of an n-th root of unity is equal to zero unless the root itself is unity, in which case n is attained.

Note that it follows from equation (2.4) that if all the Fourier coefficients beyond a certain range should vanish, say $c_k = 0$ for $|k| > M$, then the same is true for the averages S(k,t), i.e. $S(k,t) = 0$ for $k > M$.

We now introduce the Möbius function $\mu(n)$ which is given by

$$\mu(1) = 1$$

$$\mu(n) = (-1)^s \quad \text{if } n = \prod_{i=1}^{s} p_i, \ p_i \neq p_j \text{ for } i \neq j$$

$$\mu(n) = \begin{cases} 0 & \text{if } p^2 \text{ divides } n \text{ for some prime } p. \end{cases}$$

The sum over all the divisors of an integer n of the Möbius function applied to the divisors can be expressed in terms of the Kronecker delta function δ_{mn} [10,11]:

$$\sum_{m \mid n} \mu(m) = \delta_{1n} \tag{2.5}$$

This formula will enable us to derive the AFT, as follows. Fix a positive integer k. Then

$$\begin{aligned} \sum_{m=1}^{\infty} \mu(m)S(mk,t) &= \sum_{m=1}^{\infty} \mu(m) \sum_{p=1}^{\infty} (c_{mkp}e^{2\pi mkpit} + c_{-mkp}e^{-2\pi mkpit}) \\ &= \sum_{q=1}^{\infty} \left\{ \sum_{\substack{m=1 \\ m \mid q}}^{\infty} \mu(m) \right\} (c_{kq}e^{2\pi kqti} + c_{-kq}e^{-2\pi kqti}) \\ &= c_k e^{2\pi kti} + c_{-k}e^{-2\pi kti} \end{aligned} \tag{2.6}$$

From this we see that we can determine the coefficients $c_{\pm k}$ by evaluating the averages S at certain values of t. Convenient values are $t = 0, \frac{1}{4k}$, and we obtain

$$a_k = c_k + c_{-k} = \sum_{m=1}^{\infty} \mu(m)S(mk,0) \tag{2.7.a}$$

$$b_k = i(c_k - c_{-k}) = \sum_{m=1}^{\infty} \mu(m)S(mk,\frac{1}{4k}) \tag{2.7.b}$$

These formulae give us a technique for computing the Fourier coefficients of an arbitrary complex-valued periodic function with period 1 in terms of average values of samples of the function. It is readily observed that equation (2.7) involves no multiplications, and the only divisions occur in the forming of the averages (see equation (2.3)).

2.3 REDUCTION IN THE COMPUTATIONAL REQUIREMENT OF THE AFT

Here we will use an approach based on the recently published work by Reed et al. [15]. A lemma from set theory can be used to aid in the computation of the odd Fourier coefficients, b_n, without having to use a completely different set of samples (recall from the previous section that the {S(mk,0)} samples are used for the a_k coefficients and a time shifted second set, $\{S(mk,\frac{1}{4k})\}$, are used for the b_k coefficients).

The integer set $I_+ = \{1,2,3,....\}$ can be decomposed into certain disjoint integer subsets which are needed to find the coefficients b_n. These disjoint integer subsets are given by:

$$N_k = \{ 2^k(2j+1) \mid j \in I \}, \quad \text{for } k=0,1,2, ...$$

where $I = \{0\} \cup I_+ = \{0,1,2,.....\}$ is the set of nonnegative integers.

Now, instead of fixing the time argument at 1/4k for the b_k coefficients, we can use the following:

$$b_k = (-1)^j \sum_{m=1}^{[M/k]} \mu(m)\, S(mk, \frac{1}{2^{q+2}}) \qquad (2.8)$$

where q is an integer in the range $0 \le q \le [\log_2(M)]-2$ ([y] denotes the integer part of y), and k is an integer of the form $k=2^q(2j+1)$ in the intersection of set $N_q=\{ 2^q(2j+1) \mid j \in I\}$, defined earlier, and the finite set $I_{(1,N)}= \{1,2,3,.....,N\}$.

Thus, by successively choosing integers from the sets $N_k \cap I_{(1,N)}$, for $1 \le k \le [\log_2(M)]-2$, all the Fourier coefficients b_n of f(t) can be determined.

2.4 SAMPLING INTERVALS

From formula (2.7) it follows that the Fourier coefficients can only be determined by knowledge of the values of the signal at infinitely many rational points in the interval [0,1]. Clearly some assumptions have to be made to make the method practical. We shall first assume that f(t) is band-limited. Let M be a positive integer, and assume that $c_k = 0$ for $|k| > M$. Then $S(mk,t) = 0$ for $mk > M$ (m and k are integers) and we thus only have to compute S(mk,t) for $mk \le M$ and for $t = 0$ and $t = \frac{1}{4k}$ ($0 < m \le M$ and $0 < k \le M$ such that $mk \le M$).

k	Samples
6	60
7	420
8	840
9	2520
10	2520
11	27720
12	27720
13	360360
14	360360
15	360360
16	720720
17	12252240
18	12252240
19	232792560
20	232792560
21	232792560
22	232792560
23	5354228880
24	5354228880

Table 2.1 Number of samples required

Now each S(k,0) is the average of the function g(t) at the k equally spaced points $0, \frac{1}{k}, \frac{2}{k}, \ldots, \frac{k-1}{k}$. In addition S(mk,1/4k) is the average over the points $\frac{1}{4k}, \frac{m+4}{4mk}, \ldots, \frac{m-4}{4mk}$ where $1 \le m \le M$; $1 \le k \le M$; and $mk \le M$. Since we need to sample the signal at these points for all values of mk, we quickly run into an enormous number of points at which to sample. For instance, to compute S(k,0) for $0 < k \le M$ requires sampling at all of the Farey fractions [20] of order M. These are the reduced fractions j/k with $0 \le j \le k$ and $0 < k \le M$. They number $1 + \sum_{r=1}^{M} \phi(r)$ in quantity; here $\phi(r)$ is the Euler totient function. In addition the computation of $S(mk, \frac{1}{4k})$ would require another set of samples of roughly the same order of magnitude.

If uniform sampling is required then we would need to sample with a frequency equal to 4 times the least common multiple of the integers 1, 2, ..., M. For the computation of

the first 24 Fourier coefficients, namely $c_{\pm k}$ for $k \leq 12$, we would require 27720 uniformly distributed sample points to compute S(k,0) for $k \leq 12$ and approximately another 27720 such points for $S(mk,\frac{1}{4k})$. Table 2.1 gives the number of samples required for computing S(k,0) for $6 \leq k \leq 24$ based on a uniform sampling rate. The large numbers of points indicated in table 2.1 are a consequence of the uniform sampling rate, and it is important to realize that not all of these points are needed to compute k Fourier coefficients. As M approaches infinity, the number of Farey fractions asymptotically approaches $\frac{3M^2}{\pi^2}$ ([21], p. 77), which makes clear the need for some kind of interpolation. The requirement of using all of the Farey fractions of order M is, of course, based upon the supposition of uniform sampling.

As an example, we consider computing the AFT ($c_{\pm k}$ for $k \leq 6$) of the function $f(x) = -30x^4 + 60x^3 - 30x^2 + 1$ based on different uniform sampling rates. We initially compute the AFT using linear interpolation (this procedure is explained in the next section) starting with 6 samples per unit interval. We then increase the sampling rate until we reach a sampling rate above which the accuracy of the AFT does not change. This point of "saturation" is the uniform sampling rate required to compute the exact AFT. Table 2.2 shows the results of the simulation. The numbers shown at the top of each column are the number of samples of the signal per unit interval. The last column contains the true coefficients of f(x), namely, $a_k = 90/(\pi k)^4$ and $b_k = -90/(\pi k)^3$.

It is evident from the results of the simulation that in order to compute 6 Fourier coefficients, the number of samples per unit interval required to

compute the exact imaginary component of the Fourier transform is 240, which is 4 times the number required to compute the exact real component of the Fourier transform. Therefore, we conclude from this example that in order to compute the exact AFT for a given transform size, k, we need to sample the signal at 4 times the number shown in table 2.1 (this is, of course, based on uniform sampling rate).

In the next section, we introduce interpolation schemes to overcome the problem of requiring excessive sampling of the signal.

2.5 DELTA MODULATION APPROACH

The following discussion is a summary from [18]. In [5,6] the authors point out that the method may indeed be useful with delta modulation (DM) sampling systems. In such sampling systems there is a necessity for considerable oversampling of the signal in order that incremental variations of the samples remain within a single bit. Within reason, the DM approach seems to be ideally suited to the requirements of a uniformly sampled AFT algorithm, and we may be able to compute the averages with some form of difference engine.

It might be appropriate at this point to introduce some terminology for the different sampling points that we will be referring to in the following discussion. We can identify three types of sampling point (there will be a mapping of subsets of some onto others).

1 *Samplets:* These are the DM incremental valued samples ($|\text{increment}| \leq 1$ bit).

2 *Farey samples:* These correspond to the sampled value of the function at the Farey fraction intervals.

Real component of the AFT

k	/6	/12	/18	/36
1	0.88518453	0.91921252	0.92213082	0.92337388
2	0.04552472	0.05782205	0.05638462	0.05782205
3	0.01157451	0.01157451	0.01157451	0.01157451
4	0.01620394	0.00390661	0.00534403	0.00390661
5	0.04074097	0.00671297	0.00379467	0.00255162
6	0.00077170	0.00077170	0.00077170	0.00077170

k	/60	/120	/240	True
1	0.92432499	0.92432499	0.92432499	0.9239384
2	0.05782205	0.05782205	0.05782205	0.0577461
3	0.01157451	0.01157451	0.01157451	0.0114066
4	0.00390661	0.00390661	0.00390661	0.0036091
5	0.00160050	0.00160050	0.00160050	0.0014783
6	0.00077170	0.00077170	0.00077170	0.0007129

Imaginary component of the AFT

k	/6	/12	/18	/36
1	-2.59953761	-2.90202618	-2.84855413	-2.89292765
2	-0.20910484	-0.31254822	-0.34713697	-0.35877335
3	-0.00000006	-0.10416687	-0.09259284	-0.10416687
4	-0.05092621	-0.02647567	-0.04180956	-0.04260445
5	-0.09953725	-0.01464117	-0.02181613	-0.02373981
6	-0.00077188	0.00004834	-0.00780141	-0.01159680

k	/60	/120	/240	True
1	-2.89407969	-2.89407969	-2.89407969	-2.9026381
2	-0.36246485	-0.36454207	-0.36454207	-0.3628297
3	-0.10416687	-0.10416687	-0.10416687	-0.1075051
4	-0.04353797	-0.04395914	-0.04415858	-0.0453537
5	-0.02258754	-0.02258754	-0.02258754	-0.0232211
6	-0.01253474	-0.01306260	-0.01306260	-0.0134381

Table 2.2 An example showing the sampling rate requirement for exact AFT computation.

3 *Nyquist samples:* These correspond to the uniform samples taken at two-times the Nyquist frequency.

How do we determine what is a reasonable number of samplets in an interval? A pessimistic bound can be established using the assumption of f(t) having a Nyquist rate frequency component that is of maximum amplitude. This implies that the entire dynamic range may be traversed between adjacent Nyquist samples and this sets the samplet rate at 2^B times the Nyquist sample rate, where B is the number of bits in the dynamic range. For k Fourier coefficients, we assume that the Nyquist rate component is of the form $2^B e^{2pkit}$ which requires that 2k Nyquist samples be used to avoid aliasing. The total number of samplets required is therefore $k2^{B+1}$. Using this number of samples for a DM implementation we obtain a bound on k in Table 2.3 where the Farey sample requirement for uniform sampling (from Table 2.1) approximates the pessimistic samplet bound (we take the value of k that gives a lower sampling requirement than that found from Table 2.1). It seems reasonable to take the lower value, since our analysis is based on a rather pessimistic assumption of the spectral content of f(t).

B	8	12	16	20	24
k	10	12	16	18	22

Table 2.3 Bound on k for a DM implementation.

It is clear that we need to modify the uniformly sampled technique in order to obtain more Fourier coefficients than those indicated in Table 2.3. Our modification will be based on uniformly sampling at a rate no greater than that required by the pessimistic assumption of the spectral content of f(t).

2.6 NEAREST NEIGHBOR AND LINEAR INTERPOLATION

To avoid the necessity of excessive sampling of the signal, we have investigated the possibilities of both zeroth order (nearest neighbor) and linear interpolation [18]. This is a parallel investigation to that presented in [1]. To fix matters, let us assume that we are given a complex-valued signal g(t) with period 1 and mean zero, i.e. $\int_0^1 g(t)dt = 0$. We shall remove this restriction later. Let us also assume that the Fourier coefficients c_k of g(t) vanish for $|k| > M$, so that computation of the coefficients can be effected by evaluating the sums S(k,0) and $S(mk,\frac{1}{4k})$ for $0 < m \leq M$; $0 < k \leq M$; and $mk \leq M$. To this end we shall use nearest neighbor interpolation as the closest sample to the values, and linear interpolation between the values, of the signal g(t) sampled at the set of points $\{\frac{j}{N} : 0 \leq j < N\}$, where N is the number of samples per unit interval. First we state the relevant formulae that define linear interpolation

Linear interpolation:

Definition. For x a real number let [x] denote floor(x), i.e. the greatest integer ≤ x. Let {x} denote frac(x), i.e. the fractional part of x, so that x = [x]+ {x}.

Proposition. If $0 < x < 1$ then the integer k such that $\frac{k}{N} \leq x < \frac{k+1}{N}$ is given by $k = [Nx]$.

Proposition. If $0 < x < 1$ then the value of g(x) obtained through linear interpolation of the values $\{ g(0), g(\frac{1}{N}), \ldots, g(\frac{N-1}{N}) \}$ is given by

$$g(x) = (1 - \{Nx\})g_k + \{Nx\}g_{k+1} \qquad (2.9)$$

where k is given in the previous proposition and g_k denotes $g(\frac{k}{N})$. Note that by using linear interpolation we have introduced some multiplications into the scheme that were not there before. These multiplications arise from the coefficients {Nx} and 1-{Nx} in the proposition above. The AFT will, in the case of linear interpolation only involve multiplications where one factor is an integer in the range [1,M], in addition to the division by n that occurs in the definition of S(n,t).

Zero-order interpolation:

We will define zero order interpolation as follows:

$$g(\frac{i}{j}) \leftarrow g(\frac{m}{N}): \ | \frac{i}{j} - \frac{m}{N} | \leq | \frac{i}{j} - \frac{n}{N} |, n \neq m, \ n, m \in [0,N) \qquad (2.10)$$

Here $g(\frac{i}{j})$ is the Farey sample and $g(\frac{m}{N})$ is the closest available sample at a sampling rate of $\frac{1}{N}$. (The arrow in equation (2.10) indicates a replacement of the quantity on the left hand side of the arrow with the quantity on the right hand side of the arrow).

The authors in [15] developed an error analysis of the AFT method for computing Fourier coefficients using zero-order and linear interpolation procedures. The error bound for zero-order interpolation is given by

$$E[e^2] < \frac{\Delta^2}{12} \{ (\ln M) (\frac{1}{3} \ln M + \frac{2}{3} + a) + a\gamma \} R_{AA}^{(2)}(0) \qquad (2.11)$$

and for linear interpolation, the error bound is

$$E[e^2] < \frac{\Delta^2}{(12)^2} \{ (\ln M) (\frac{1}{3} \ln M + \frac{2}{3} + a) + a\gamma \} R_{AA}^{(4)}(0) \qquad (2.12)$$

S_1	g_0
S_2	$\frac{1}{2}(g_0 + g_{30})$
S_3	$\frac{1}{3}(g_0 + g_{20} + g_{40})$
S_4	$\frac{1}{4}(g_0 + g_{15} + g_{30} + g_{45})$
S_5	$\frac{1}{5}(g_0 + g_{12} + g_{24} + g_{36} + g_{48})$
S_6	$\frac{1}{6}(g_0 + g_{10} + g_{20} + g_{30} + g_{40} + g_{50})$
S_7	$\frac{1}{7}(g_0 + g_9 + g_{17} + g_{26} + g_{34} + g_{43} + g_{51})$
S_8	$\frac{1}{8}(g_0 + g_7 + g_{15} + g_{23} + g_{30} + g_{37} + g_{45} + g_{53})$
S_9	$\frac{1}{9}(g_0 + g_7 + g_{13} + g_{20} + g_{27} + g_{33} + g_{40} + g_{47} + g_{53})$
S_{10}	$\frac{1}{10}(g_0 + g_6 + g_{12} + g_{18} + g_{24} + g_{30} + g_{36} + g_{42} + g_{48} + g_{54})$
S_{11}	$\frac{1}{11}(g_0 + g_5 + g_{11} + g_{16} + g_{22} + g_{27} + g_{33} + g_{38} + g_{44} + g_{49} + g_{55})$
S_{12}	$\frac{1}{12}(g_0 + g_5 + g_{10} + g_{15} + g_{20} + g_{25} + g_{30} + g_{35} + g_{40} + g_{45} + g_{50} + g_{55})$

Table 2.4 Zero order interpolation sums for S values

where $E[e^2]$ is mean square error over a period, M is output transform size, $a = \frac{1}{4}\ln 4 + \frac{1}{18}\ln 9 + \frac{1}{36}\ln 18 + 1$, $\gamma \approx 0.577$ (Euler's constant), Δ is the sampling interval, and $R_{AA}^{(2)}$ and $R_{AA}^{(4)}$ are, respectively, the 2nd-order and 4th-order derivative of the autocorrelation function of f(t) (recall $f(t) = g(t) + c_0$). The sampling errors, $\varepsilon_n = g(\frac{k}{n} + t) - \tilde{g}(\frac{k}{n} + t)$, where $\tilde{g}(t)$ is the reconstructed waveform with error, are assumed to be independent, identically distributed, zero-mean, random variables for different values for k and n.

We now introduce some tables to give the interpolants to the various values of S(k,0) and $S(mk, \frac{1}{4k})$ for $0 < M \leq 12$ and N=60, based on the results of section

$B_{1\,1}$	g_{15}
$B_{1\,2}$	$\frac{1}{2}(g_{15} + g_{45})$
$B_{1\,3}$	$\frac{1}{3}(g_{15} + g_{35} + g_{55})$
$B_{1\,5}$	$\frac{1}{5}(g_3 + g_{15} + g_{27} + g_{39} + g_{51})$
$B_{1\,6}$	$\frac{1}{6}(g_5 + g_{15} + g_{25} + g_{35} + g_{45} + g_{55})$
$B_{1\,7}$	$\frac{1}{7}(g_6 + g_{15} + g_{24} + g_{32} + g_{41} + g_{49} + g_{58})$
$B_{1\,9}$	$\frac{1}{9}(g_2 + g_8 + g_{15} + g_{22} + g_{28} + g_{35} + g_{42} + g_{48} + g_{55})$
$B_{1\,10}$	$\frac{1}{10}(g_3 + g_9 + g_{15} + g_{21} + g_{27} + g_{33} + g_{39} + g_{45} + g_{51} + g_{57})$
$B_{1\,11}$	$\frac{1}{11}(g_4 + g_9 + g_{15} + g_{20} + g_{26} + g_{31} + g_{37} + g_{42} + g_{48} + g_{53} + g_{59})$
$B_{2\,2}$	$\frac{1}{2}(g_7 + g_{37})$
$B_{2\,4}$	$\frac{1}{4}(g_7 + g_{22} + g_{37} + g_{52})$
$B_{2\,6}$	$\frac{1}{6}(g_7 + g_{17} + g_{27} + g_{37} + g_{47} + g_{57})$
$B_{2\,10}$	$\frac{1}{10}(g_2 + g_8 + g_{14} + g_{20} + g_{26} + g_{32} + g_{38} + g_{44} + g_{50} + g_{56})$
$B_{2\,12}$	$\frac{1}{12}(g_2 + g_7 + g_{12} + g_{17} + g_{22} + g_{27} + g_{32} + g_{37} + g_{42} + g_{47} + g_{52} + g_{57})$
$B_{4\,4}$	$\frac{1}{4}(g_4 + g_{19} + g_{34} + g_{49})$
$B_{4\,8}$	$\frac{1}{8}(g_4 + g_{11} + g_{19} + g_{26} + g_{34} + g_{41} + g_{49} + g_{56})$
$B_{4\,12}$	$\frac{1}{12}(g_4 + g_9 + g_{14} + g_{19} + g_{24} + g_{29} + g_{34} + g_{39} + g_{44} + g_{49} + g_{54} + g_{59})$
$B_{8\,8}$	$\frac{1}{8}(g_2 + g_9 + g_{17} + g_{24} + g_{32} + g_{39} + g_{47} + g_{54})$

Table 2.5 Zero order interpolation sums for B values

2.3, and for both zeroth order (tables 2.4 and 2.5) and linear (tables 2.6 and 2.7) interpolation. In table 2.4 we have the formulae for the computation of S(k,0), $0 < k \leq 12$. They are represented by the symbol S_k. Note that in the equations of table 2.4 there are no multiplications and only 11 divisions with the integers 2 through 12 (inclusive) used as divisors. In table 2.5 are the formulae for $S(\imath nk, \frac{1}{4k})$, with conclusions to be drawn as to the number of multiplications and divisions, namely zero multiplications and 17 integer

S1	g_0
S_2	$\frac{1}{2}(g_0 + g_{30})$
S_3	$\frac{1}{3}(g_0 + g_{20} + g_{40})$
S_4	$\frac{1}{4}(g_0 + g_{15} + g_{30} + g_{45})$
S_5	$\frac{1}{5}(g_0 + g_{12} + g_{24} + g_{36} + g_{48})$
S_6	$\frac{1}{6}(g_0 + g_{10} + g_{20} + g_{30} + g_{40} + g_{50})$
S_7	$\frac{1}{49}(7g_0+(g_{18}+g_{42})+2(g_{25}+g_{35})+3(g_8+g_{52})+4(g_9+g_{51})+5(g_{26}+g_{34})+6(g_{17}+g_{43}))$
S_8	$\frac{1}{16}(2(g_0+g_{15}+g_{30}+g_{45})+g_7+g_8+g_{22}+g_{23}+g_{37}+g_{38}+g_{52}+g_{53})$
S_9	$\frac{1}{27}(3(g_0+g_{20}+g_{40})+(g_6+g_{14}+g_{26}+g_{34}+g_{46}+g_{54})+2(g_7+g_{13}+g_{27}+g_{33}+g_{47}+g_{53}))$
S_{10}	$\frac{1}{10}(g_0 + g_6 + g_{12} + g_{18} + g_{24} + g_{30} + g_{36} + g_{42} + g_{48} + g_{54})$
S_{11}	$\frac{1}{121}(11g_0+(g_{10}+g_{50})+2(g_{21}+g_{39})+3(g_{28}+g_{32})+4(g_{17}+g_{43})+5(g_6+g_{54})+6(g_5+g_{55})+7(g_{16}+g_{44})+8(g_{27}+g_{33})+9(g_{22}+g_{38})+10(g_{11}+g_{49}))$
S_{12}	$\frac{1}{12}(g_0 + g_5 + g_{10} + g_{15} + g_{20} + g_{25} + g_{30} + g_{35} + g_{40} + g_{45} + g_{50} + g_{55})$

Table 2.6 Linear interpolatien sums for S values

divisions. We have used the term $B_{k\ mk} = S(mk,\frac{1}{4k})$ in table 2.5 (the first subscript k refers to the k-th coefficient and the second subscript mk refers to the order of the average, i.e., mk refers to the first argument, k refers to the second argument of $S(mk,\frac{1}{4k})$. In tables 2.6 and 2.7 are the formulae for the computations of $S_k = S(k,0)$ and $B_{k\ mk} = S(mk,\frac{1}{4k})$, but this time with linear interpolation as indicated by the propositions of this section. In table 2.6 there are 19 multiplications and, again, eleven divisions. This time the divisors can involve integers up to 121 in value. In table 2.7 there are 36 multiplications and 17 divisions. The comparison between the zeroth and first order

$B_{1\,1}$	g_{15}
$B_{1\,2}$	$\frac{1}{2}(g_{15} + g_{45})$
$B_{1\,3}$	$\frac{1}{3}(g_{15} + g_{35} + g_{55})$
$B_{1\,5}$	$\frac{1}{5}(g_3 + g_{15} + g_{27} + g_{39} + g_{51})$
$B_{1\,6}$	$\frac{1}{6}(g_5 + g_{15} + g_{25} + g_{35} + g_{45} + g_{55})$
$B_{1\,7}$	$\frac{1}{49}(7g_{15} + (g_{33}+g_{57}) + 2(g_{40}+g_{50}) + 3(g_7+g_{23}) + 4(g_6+g_{24}) + 5(g_{41}+g_{49}) + 6(g_{32}+g_{58}))$
$B_{1\,9}$	$\frac{1}{27}((g_1+g_9+g_{21}+g_{29}+g_{41}+g_{49}) + 2(g_2+g_8+g_{22}+g_{28}+g_{42}+g_{48}) + 3(g_{15}+g_{35}+g_{55}))$
$B_{1\,10}$	$\frac{1}{10}(g_3 + g_9 + g_{15} + g_{21} + g_{27} + g_{33} + g_{39} + g_{45} + g_{51} + g_{57})$
$B_{1\,11}$	$\frac{1}{121}(11g_{15} + (g_5+g_{25}) + 2(g_{36}+g_{54}) + 3(g_{43}+g_{47}) + 4(g_{32}+g_{58}) + 5(g_9+g_{21}) + 6(g_{10}+g_{20}) + 7(g_{31}+g_{59}) + 8(g_{42}+g_{48}) + 9(g_{37}+g_{53}) + 10(g_4+g_{26}))$
$B_{2\,2}$	$\frac{1}{4}(g_7+g_8+g_{37}+g_{38})$
$B_{2\,4}$	$\frac{1}{8}(g_7 + g_8 + g_{22} + g_{23} + g_{37} + g_{38} + g_{52} + g_{53})$
$B_{2\,6}$	$\frac{1}{12}(g_7 + g_8 + g_{17} + g_{18} + g_{27} + g_{28} + g_{37} + g_{38} + g_{47} + g_{48} + g_{57} + g_{58})$
$B_{2\,10}$	$\frac{1}{20}(g_1 + g_2 + g_7 + g_8 + g_{13} + g_{14} + g_{19} + g_{20} + g_{25} + g_{26} + g_{31} + g_{32} + g_{37} + g_{38} + g_{43} + g_{44} + g_{49} + g_{50} + g_{55} + g_{56})$
$B_{2\,12}$	$\frac{1}{24}(g_2 + g_3 + g_7 + g_8 + g_{12} + g_{13} + g_{17} + g_{18} + g_{22} + g_{23} + g_{27} + g_{28} + g_{32} + g_{33} + g_{37} + g_{38} + g_{42} + g_{43} + g_{47} + g_{48} + g_{52} + g_{53} + g_{57} + g_{58})$
$B_{4\,4}$	$\frac{1}{16}((g_3 + g_{33}) + 3(g_4 + g_{34}) + (g_{18} + g_{48}) + 3(g_{19} + g_{49}))$
$B_{4\,8}$	$\frac{1}{32}((g_3 + g_{33}) + 3(g_4 + g_{34}) + (g_{18} + g_{48}) + 3(g_{19} + g_{49}) + 3(g_{11} + g_{41}) + (g_{12} + g_{42}) + 3(g_{26} + g_{56}) + (g_{27} + g_{57}))$
$B_{4\,12}$	$\frac{1}{48}((g_3 + g_{33}) + 3(g_4 + g_{34}) + (g_8 + g_{38}) + 3(g_9 + g_{39}) + (g_{13} + g_{43}) + 3(g_{14} + g_{44}) + (g_{18} + g_{48}) + 3(g_{19} + g_{49}) + (g_{23} + g_{53}) + 3(g_{24} + g_{54}) + (g_{28} + g_{58}) + 3(g_{29} + g_{59}))$
$B_{8\,8}$	$\frac{1}{64}((g_1 + g_{31}) + 7(g_2 + g_{32}) + 5(g_9 + g_{39}) + 3(g_{10} + g_{40}) + (g_{16} + g_{46}) + 7(g_{17} + g_{47}) + 5(g_{24} + g_{54}) + 3(g_{25} + g_{55}))$

Table 2.7 Linear interpolation sums for B values

interpolation schemes for the AFT is summed up in table 2.8. From the computational point of view, the zero-order interpolation scheme only requires divisions versus the requirement of both multiplication and divisions for the linear interpolation case. The divisions for zero order interpolation are a maximum of 4 bits (i.e. only 4 'one' bits in the coefficient) and this can be used to provide simplified binary division algorithms either in special software routines or in hardware.

	Zeroth Order		First Order	
	S_k	B_k mk	S_k	B_k mk
Multiplications	0	0	19	36
Divisions	11	17	11	17

Table 2.8 Computational requirements

2.7 REMOVING THE ZERO MEAN ASSUMPTION

It was implicitly assumed from equation (2.2) that the function g(t) can be decomposed into a set of Fourier coefficients that do not include the dc-term. In other words, we assume that the sequence has zero mean over the interval [0,1). If we can relax this restriction, then it is possible to compute the Fourier coefficients of a signal with any arbitrary mean value. In order to do this we present the following theorem [18].

Theorem I

Let g(t) be an arbitrary periodic integrable function, period 1, with $g_0 = g(0)$ and $\bar{g} = \int_0^1 g(t)dt$. Define $f(t) = g(t) - \bar{g}$, so that $\bar{f} = 0$, and $h(t) = g(t) - g_0$, so that $h(0) = 0$. Then the Fourier coefficients of g(t) are given by:

$$a_k(g) = \sum_{m=1}^{\infty} \mu(m)[S(h,mk,0) + g_0 - \bar{g}] \qquad (2.13.a)$$

$$b_k(g) = \sum_{m=1}^{\infty} \mu(m)[S(h,mk,1/4k) + g_0 - \bar{g}] \qquad (2.13.b)$$

Proof

We have $S(f,k,0) = S(g - \bar{g},k,0) = S(g,k,0) - S(\bar{g},k,0) = S(g,k,0) - \bar{g}$. Next, for $k \geq 1$,

$$a_k(g) = a_k(f) = \sum_{m=1}^{\infty} \mu(m)S(f,mk,0) = \sum_{m=1}^{\infty} \mu(m)[S(g,mk,0) - \bar{g}]$$

Since $h(t) = g(t) - g_0$, we have $S(g,k,0) = S(h+g_0,k,0) = S(h,k,0) + g_0$, hence the result:

$$a_k(g) = a_k(f) = \sum_{m=1}^{\infty} \mu(m)S(f,mk,0) = \sum_{m=1}^{\infty} \mu(m)[S(g,mk,0)-\bar{g}]$$

$$= \sum_{m=1}^{\infty} \mu(m)[S(h,mk,0) + g_0 - \bar{g}]$$

Similarly,

$$b_k(g) = \sum_{m=1}^{\infty} \mu(m)[S(h,mk,1/4k) + g_0 - \bar{g}].$$

Therefore, Equation (2.11) defines the AFT algorithm for computing the Fourier coefficients of a signal with any arbitrary mean value. Appendix A contains a FORTRAN code for computing the AFT, equation (2.13), of any complex-valued periodic function using zero order and linear interpolation procedures.

It is important to note that the inclusion of the quantity $(g_0 - \bar{g})$ for each coefficient can be delayed, from an implementation point of view, until the final summation for the Fourier coefficients.

2.8 SOME EXAMPLES

We have prepared some numerical examples to indicate the kind of accuracy one may expect in computing Fourier coefficients with such a technique as the AFT. The following examples suggest that the accuracy of the method is of a similar order of magnitude as one can obtain through the DFT. For the purpose of comparing the AFT with the traditional computation scheme involving the numerical approximation of trigonometric integrals with trapezoidal rule, we considered functions from four classes: fourth-order polynomial; linear; sinusoidal; and gaussian. In each case we computed the AFT over 60 points of interpolation, equally spaced over the interval [0,1]. We computed the Fourier coefficients c_k for $|k| \leq 30$ using zeroth order (nearest neighbor) and first order (linear) interpolation. The Fourier coefficients of these functions are given in Appendix B. In addition we computed, for each function, the inverse Fourier transform (using the IDFT with the trapezoidal rule) to transform the AFT back to original domain, which we give below.

In the first figure, Fig 2.1, we used the polynomial function [18] $y = 1 - 30\,t + 60\,t^2 - 30\,t^4$ (in the figure, $t = k/60$). This function is, of course, not band-limited, so it offers the potential for aliasing. On the other hand it does behave somewhat like a periodic function in that $y(0) = y(1)$ and $y'(0) = y'(1)$ (together with $\int_0^1 y(t)\,dt = 0$), so that convergence of the Fourier series is rapid. All three methods of interpolation (AFT with zero order interpolation, AFT with linear interpolation, and DFT using trapezoidal rule) showed good results. For the second function, shown in Fig 2.2, we used the deceptively simple straight line $y = t$, a function which is not periodic and whose Fourier series converges quite slowly (the coefficients c_k decrease in magnitude like $\frac{1}{k}$).

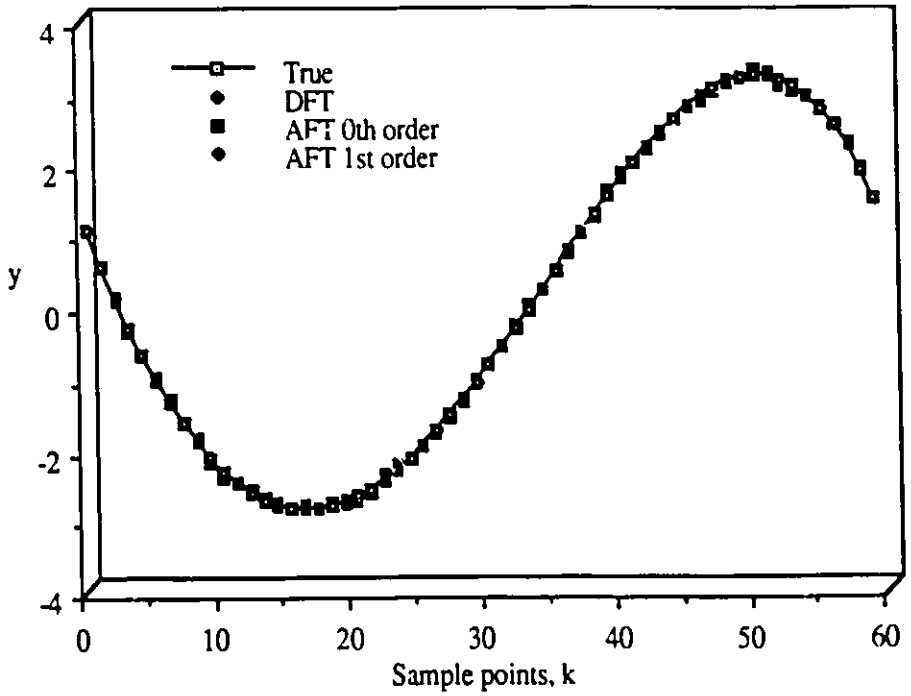


Fig 2.1: $y = 1 - 30t + 60t^2 - 30t^4$

All three methods seemed essentially equal in the degree of approximation. The exceptional points to the left and right of the graphs are explained through the lack of periodicity of the function $y = t$. In Fig 2.3 we have approximated the symmetric gaussian function $y = \exp[(t - 1/2)^2/0.03]$, which is clearly periodic, since $y(0) = y(1)$. All three approximations were good. In Fig 2.4 we have approximated the function $y = \sin(6\pi t) + \sin(18\pi t) + \sin(34\pi t)$, which is clearly not periodic. All three approximations were good; the AFT does, however, have noticeable degradation; the 0th order interpolation scheme shows more degradation. This indicates that the AFT with 0th order interpolation is susceptible to degradation when the energy of the signal is concentrated in a small finite number of harmonics. However, both schemes

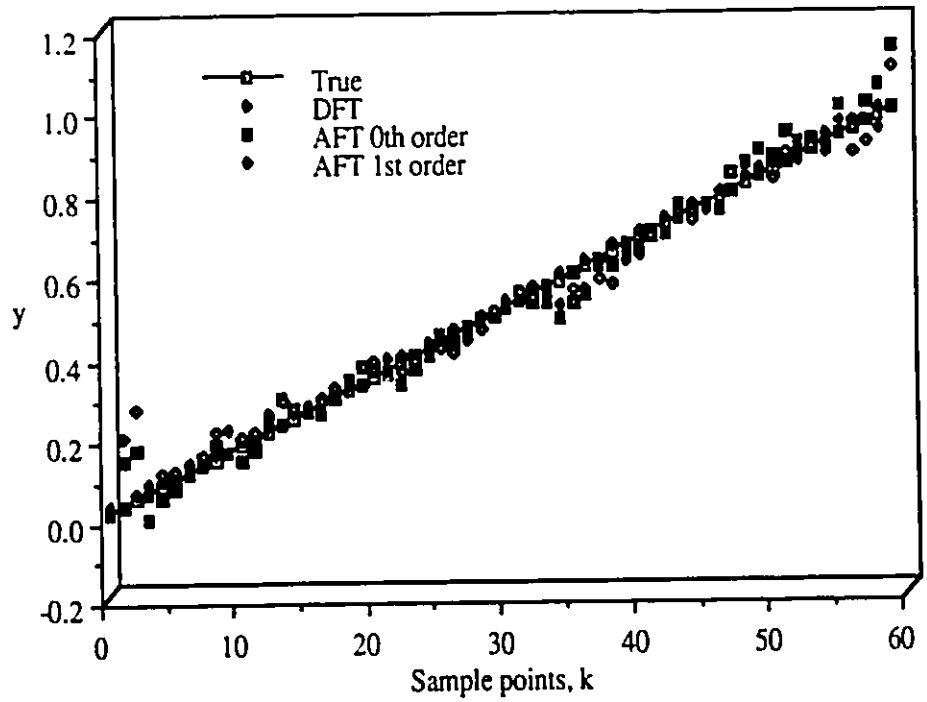


Fig 2.2 : y = t

are quite close. The DFT calculation is more accurate than the AFT, and is more suited for this type of functions. This is because the basis functions for the DFT calculation are trigonometric functions.

Conclusions to be drawn from these graphs are that both versions of the AFT seem to be as accurate as the DFT for computing Fourier coefficients. Computation of the AFT will be radically faster than for the DFT, at least insofar as ordinary binary arithmetic is used. From the graphs shown here it seems that the AFT with linear interpolation offers no significant advantage over the AFT with zeroth order interpolation, and yet it comes with a price of several multiplications.

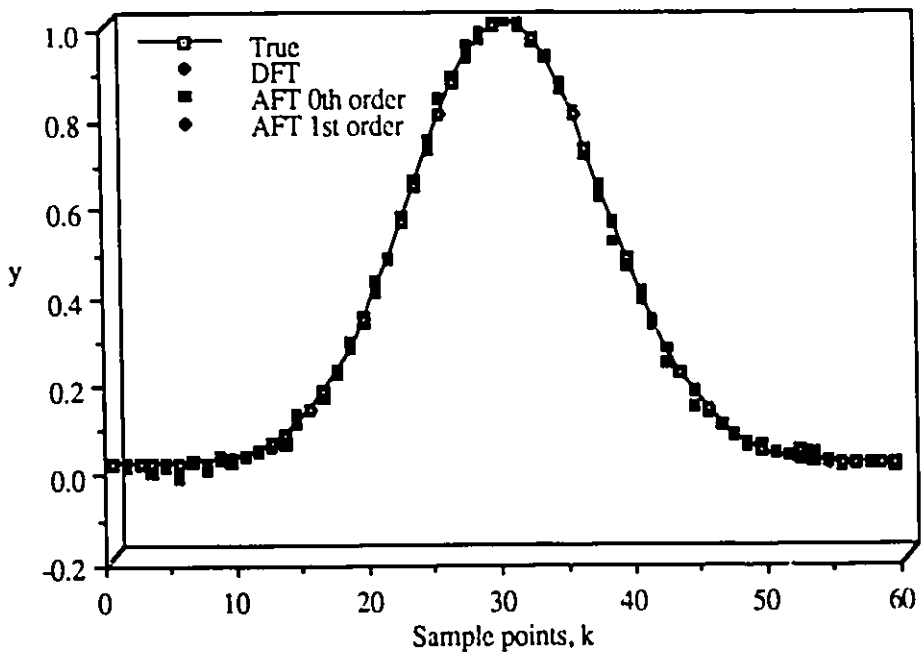


Fig 2.3 : $y = \exp[(t-1/2)^2/0.03]$

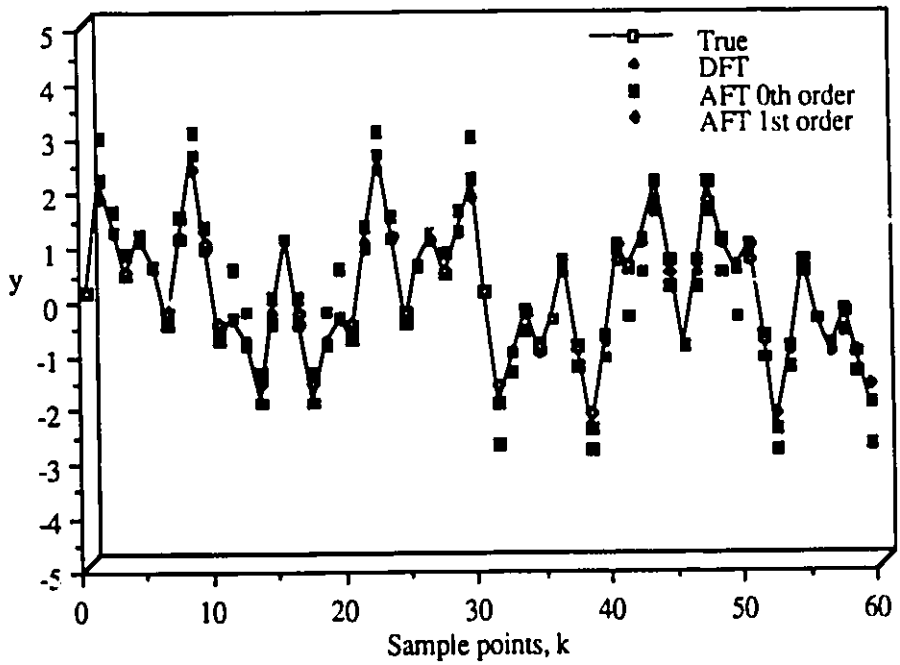


Fig 2.4 : $y = \sin(6\pi t) + \sin(18\pi t) + \sin(34\pi t)$

2.9 COMPUTATIONAL COMPLEXITY

In this section, we consider the computational complexity of the original AFT algorithm, equation (2.7), with respect to the familiar FFT algorithm. We point out that the FFT is a fast implementation of the Discrete Fourier Transform (DFT) and consequently a considerable reduction in arithmetic operations to compute the DFT is accomplished by it . Moreover, the AFT is considered to be, up to now, at the stage of brute-force computation, as was the DFT before the invention of the FFT algorithms.

The total number of arithmetic operations required in an algorithm is the criterion we consider here for computational complexity. We summarize this aspect in table 2.9 (see Appendix C for details), where the number of arithmetic operations in terms of divisions (multiplications for FFT and DFT) and additions is listed as a function of M, the output transform size, for the AFT, FFT [22], and DFT [22] algorithms. The FFT algorithm is based on radix-2 factorizations.

Algorithm	Original AFT	Radix-2 FFT	DFT
# of additions	$O(M^2)$	$O(M \log_2 M)$	$O(M^2)$
# of multiplications	$O(M)$	$O(M \log_2 M)$	$O(M^2)$

Table 2.9 : Computational complexity for original AFT, radix-2 FFT, and DFT.

Note that in the FFT and DFT algorithms all arithmetic operations are complex, whereas in the AFT algorithm the additions are complex and the divisions are by real numbers. Thus, for a complex-valued signal, each

complex multiplication is equivalent to 4 real multiplications and 2 real additions. Therefore, according to table 2.9, in terms of the total number of multiplications (or divisions), the AFT algorithm will be radically faster for large M. In terms of the number of additions, the AFT involves $O(M^2)$ operations while the FFT involves $O(M\log_2 M)$ operations. However, additions are not as costly as multiplications in an IC realization.

It is also important to note that while the multiplications involved in the FFT and DFT algorithms are by complex twiddle factors performed at different stages of the computation, the divisions involved in the AFT algorithm are by real integers performed at one stage of the computation. Hence, it is hoped that the AFT algorithm may offer accurate, high speed Fourier analysis and synthesis.

2.10 SUMMARY

In this chapter we have considered techniques for computation of the Arithmetic Fourier transform. Our objective has been to show how the demand for oversampling of the signal required by the exact AFT is avoided by using approximate computational techniques.

We derived the AFT algorithm, as it is not found in any of the usual textbooks on Fourier analysis. We outlined the problems of excessive sampling of the signal needed to compute the exact AFT. Examples were used to depict the problem. We described two approximate computational techniques, zero order interpolation and linear interpolation, with examples in detail.

We have discussed the suitabitiy of delta modulation (DM) to the requirement of oversampling of the input signal needed to compute the

exact AFT. We found that we need to modify the uniformly sampled technique in order to obtain more Fourier coefficients that those indicated in table 2.3.

We have considered some examples to show the kind of accuracy the AFT may offer in computing Fourier coefficients, in comparison to the standard technique of computation using the discrete Fourier transform. It was shown that the accuracy of the AFT is of a similar order of magnitude as that of the DFT.

We have considered the computational complexity and the direct evaluation of the AFT algorithm. We compared the AFT with the familiar radix-2 FFT and DFT algorithms in terms of the number of arithmetic operations. We found out that in terms of multiplications (or divisions for the AFT) the AFT will be radically faster for large transform sizes. In terms of additions, the AFT involves $O(M^2)$ operations compared to only $O(M\log_2 M)$ operations for the radix-2 FFT.

CHAPTER 3

ON IMPLEMENTING THE ARITHMETIC FOURIER TRANSFORM (AFT)

3.1 INTRODUCTION

In this chapter we will present implementation schemes for the AFT that can be translated into source code for running on conventional computer systems, or into hardware for very high speed analysis. The present chapter contains several new different VLSI implementations of the AFT algorithm. The AFT implementations are based on nearest neighbor and linear interpolation procedures discussed in the previous chapter. We start by examining the special case of analysis of symmetric functions; special computational symmetries are found which can aid in the computing process. In this part we will present two VLSI architectures, which are suitable for both interpolation procedures. In the second part, we examine the implementation of the AFT for general functions. In this part we will present three VLSI architectures— the first is suitable for implementation using both interpolation procedures and the last two are suitable for implementation using nearest neighbor interpolation procedure.

3.2 ARCHITECTURE I

We will consider here the AFT implementation [18] of zero order and linear interpolation procedures for functions which have a cosine Fourier series expansion. These functions are symmetric about the line t=1/2, so that the coefficients of the sine terms of the Fourier expansion are zero. We can perform the calculations by considering symmetries associated with computation of the averages S(n,0). We will base the discussion in this section on the 60 sample, 12 coefficient example used earlier in chapter 2 (section 5).

3.2.1 LINEAR INTERPOLATION

We will start with the linear interpolation case, since this is the more complex of the two. The sample multipliers, associated with the $\{S_i\}$, where S_i = S(i,0), are displayed in Table 3.1. The columns correspond to the samples required for each of the $\{S_i\}$, and the rows correspond to the samples. The unused sample set is removed from the table. If we neglect the g_0 sample, there is clearly a mirror symmetry about the g_{30} sample associated with the Hermitian symmetry of the roots of unity.

It is interesting that this symmetry turns up because we have not actually used these roots of unity in any of the calculations. This symmetry can be used to remove close to[1] 50% of the non-trivial multiplications required in the calculation of the $\{S_i\}$. If we now treat calculations with g_0 as special, then we can reduce the other calculations to take into account the symmetry. This reduction is shown in Table 3.2.

---

[1] There are 4 non-trivial multiplications of g_0 and 1 of g_{30} required in the computations.

Another interesting symmetry (but not apparently useful for reduction of arithmetic operations) is the fact that every vertical grouping (adjacent values) in each column adds to the g_0 multiplier on the first row of the table. Table 3.3 shows the divisors associated with each of the $\{S_i\}$.

We have isolated calculations with g_0 from the main computational structure. In order to do this we use the result of theorem I developed in section 6 of chapter 2. Table 3.4, gives the multiplier of the offset, $g_0 - \bar{g}$, for the 12 Fourier coefficients. We note that only one value has a multiplier other than ±1 and this is a multiplier of -2, which will be very easy to implement. In exchange for this, we have a saving of 3 summations compared to the original requirement to modify all 12 of the $\{S_k\}$ values.

	1	2	3	4	5	6	7	8	9	10	11	12
0	1	1	1	1	1	1	7	2	3	1	11	1
5											6	1
6									1	1	5	
7								1	2			
8							3	1				
9							4					
10						1					1	1
11											10	
12					1					1		
13									2			
14									1			
15				1				2				1
16											7	
17							6				4	
18							1			1		
19												
20			1			1			3			1
21											2	
22								1			9	
23								1				
24					1					1		
25							2					1
26							5		1			
27									2		8	
28											3	
30		1		1		1		2		1		1
32											3	
33									2		8	
34							5		1			
35							2					1
36					1					1		
37								1				
38								1			9	
39											2	
40			1			1			3			1
41												
42							1			1		
43							6				4	
44											7	
45				1				2				1
46									1			
47									2			
48					1					1		
49											10	
50						1					1	1
51							4					
52							3	1				
53								1	2			
54									1	1	5	
55											6	1

Table 3.1 Linear Interpolation Averages

	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}
g_5+g_{55}										6	1
g_6+g_{54}								1	1	5	
g_7+g_{53}							1	2			
g_8+g_{52}						3	1				
g_9+g_{51}						4					
$g_{10}+g_{50}$					1					1	1
$g_{11}+g_{49}$										10	
$g_{12}+g_{48}$				1					1		
$g_{13}+g_{47}$								2			
$g_{14}+g_{46}$								1			
$g_{15}+g_{45}$			1				2				1
$g_{16}+g_{44}$										7	
$g_{17}+g_{43}$						6				4	
$g_{18}+g_{42}$						1			1		
$g_{19}+g_{41}$											
$g_{20}+g_{40}$		1			1			3			1
$g_{21}+g_{39}$										2	
$g_{22}+g_{38}$							1			9	
$g_{23}+g_{37}$							1				
$g_{24}+g_{36}$				1					1		
$g_{25}+g_{35}$						2					1
$g_{26}+g_{34}$						5		1			
$g_{27}+g_{33}$								2		8	
$g_{28}+g_{32}$										3	
g_{30}	1		1		1		2		1		1

Table 3.2 Symmetry Reduction

S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}
1	2	3	4	5	6	49	16	27	10	121	12

Table 3.3 $\{S_i\}$ Divisors

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
-2	-1	-1	-1	0	0	1	1	1	1	1	1

Tabie 3.4 Multiplier of the offset $(g_0 - \bar{g})$

The input subtraction of g_0 from each sample only requires a single hardware subtractor in the pipeline register input. The use of the offset, $g_0 - \bar{g}$, is after the computation of the $\{S_k\}$ and clearly at a point in the computations where the mean value of the sequence of 60 samples will already have been computed.

	1	2	3	4	5	6	7	8	9	10	11	12
0	1	1	1	1	1	1	1	1	1	1	1	1
5											1	1
6										1		
7								1	1			
9							1					
10						1						1
11											1	
12					1					1		
13									1			
15				1				1				1
16											1	
17							1					
18										1		
20			1			1			1			1
22								1			1	
24					1					1		
25												1
26							1					
27									1		1	
30		1		1		1		1		1		1
33									1		1	
34							1					
35												1
36					1					1		
38								1			1	
40			1			1			1			1
42										1		
43							1					
44											1	
45				1				1				1
47									1			
48					1					1		
49											1	
50						1						1
51							1					
53								1	1			
54										1		
55											1	1

Table 3.5 Zero Order Averages

3.1.2 ZERO ORDER INTERPOLATION

Following the same search for symmetry in the zero order case, we can generate Tables 3.5 and 3.6. We note that the non-trivial multipliers associated with the linear interpolation case have now reverted to unity multipliers. The same mirror symmetry is observed in Table 3.5, with the improvement of only trivial multipliers and a reduction in the number of samples required. Table 3.6 indicates that the same computational approach can be taken for the zero order case as the linear interpolation case. Again the multipliers are unity, and the number of

	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}
g_5+g_{55}										1	1
g_6+g_{54}									1		
g_7+g_{53}							1	1			
g_9+g_{51}						1					
$g_{10}+g_{50}$					1						1
$g_{11}+g_{49}$										1	
$g_{12}+g_{48}$				1					1		
$g_{13}+g_{47}$								1			
$g_{15}+g_{45}$			1				1				1
$g_{16}+g_{44}$										1	
$g_{17}+g_{43}$						1					
$g_{18}+g_{42}$									1		
$g_{20}+g_{40}$		1			1			1			1
$g_{22}+g_{38}$							1			1	
$g_{24}+g_{36}$				1					1		
$g_{25}+g_{35}$											1
$g_{26}+g_{34}$						1					
$g_{27}+g_{33}$								1		1	
g_{30}	1		1		1		1		1		1

Table 3.6 Symmetry Reduction

S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}
1	2	3	4	5	6	7	8	9	10	11	12

Table 3.7 $\{S_i\}$ Divisors

pairwise summed samples is reduced. Table 3.7 shows the divisors associated with each of the $\{S_i\}$.

3.1.3 THE ARCHITECTURE

The block diagram of the architecture is shown in Fig 3.1. Assume that the signal samples appear at the input sequentially, g_0, g_1, etc. The first N/2 samples are fed to the left N/2-stage shift register, after subtracting g_0 from each sample, and then switched to the parallel load N/2-stage shift register via parallel load operation. The parallel load shift register is used to reverse

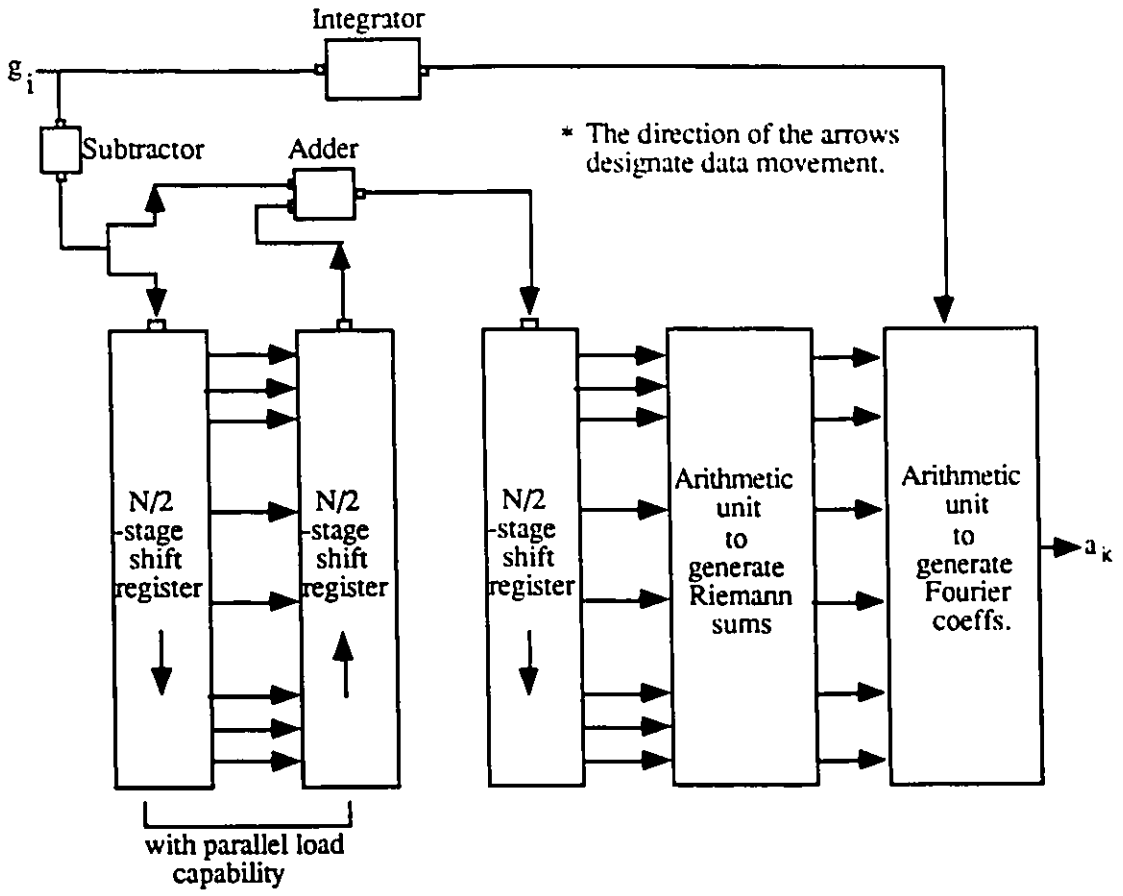


Fig 3.1 : Block diagram of architecture I.

the first half of the samples where they are summed with the latter half of the samples using the alignment shown in table 3.2 for linear interpolation procedure or table 3.6 for zero-order interpolation procedure. The output summed pairs are fed to the right N/2-stage shift register. The next action for the linear interpolation scheme is that the outputs of the right shift register are fed through (the arithmetic unit to generate the Riemann sums) a set of multipliers and summed (according to table 3.2) and a set of dividers with the divisors given by table 3.3. For the zero-order interpolation scheme, the outputs of the right shift register are fed through a set of adders (according to table 3.6) and a set of dividers with the divisors given table 3.7. The resulting

values of { S_k } are summed accordingly in the arithmetic unit to generate the AFT of the input signal. Notice that the operation described by table 3.4 can be delayed until the final summation for Fourier coefficients as shown in the diagram. The integrator (an accumulator) is used to find the mean of the signal. In both interpolation schemes, the arithmetic unit to generate the Fourier coefficients consists of a set of adders with subtract capability, a divider for the calculation of the mean, and a multiplier required for performing the calculation of table 3.4.

It is clear that the architecture just described is quite involved in data transfer and manipulation, for the actual computing of the AFT takes place only in the last two arithmetic sub-blocks. In addition, the architecture's speed is restricted by the speed of the input adder (for a non-pipelined implementation).

The verification of the correctness of this architecture as well as the following architectures have been undertaken and the results of the simulations are presented in the next chapter.

3.3 ARCHITECTURE II

The architecture discussed in the previous section seemed to be largely involved and dominated by data manipulation and transfers. The actual processing (computing) time of the AFT algorithm is relatively short with respect to the data transfer time, and therefore the processing speed is limited by the registers/adder (the front end of the architecture) performance. This feature makes architecture I not suitable for high speed applications.

In this section, the AFT implementation of zero-order and linear interpolation procedures for symmetric functions using a modified version of

the previous architecture is presented. The modified architecture is obtained by removing the right shift register/adder combination from architecture I, based on a property of the AFT algorithm. The new structure is very efficient for VLSI implementation.

3.3.1 THE ARCHITECTURE

The AFT algorithm for symmetric functions is given by the following equation:

$$a_k = \sum_{m=1}^{[M/k]} \mu(m)\, S(mk,0) \quad ; k=1,2,.....,M$$

where $S(n,0) = \frac{1}{n}\sum_{k=0}^{n-1} g(\frac{k}{n})$ is the Riemann sum of order n and μ is the Möbius function. We will now observe that each Riemann sum can be expressed as a sum of two parts, each of which is concerned with the summation of samples in one half period of the signal. That is,

For linear interpolation:

$$S(n,0) = \frac{1}{n}\{ \sum_{k=0}^{\frac{n}{2}-1} g(\frac{k}{n}) + \sum_{k=0}^{\frac{n}{2}-1} g(\frac{k}{n}+\frac{1}{2}) \}$$

$$= \frac{1}{n}\{ W_n(\frac{n}{2}, 0) + W_n(\frac{n}{2}, \frac{1}{2}) \} \qquad \text{for even n} \quad (3.1.a)$$

$$n^2 S(n,0) = \sum_{(i<N/2, i\neq 0)} \alpha_i (g_i + g_{N-i})$$

$$= \sum_{(i<N/2, i\neq 0)} \alpha_i g_i + \sum_{(i<N/2, i\neq 0)} \alpha_i g_{N-i} \qquad \text{for odd n} \quad (3.1.b)$$

For zero-order interpolation:

$$S(n,0)=\frac{1}{n}\{\sum_{k=0}^{\frac{n}{2}-1} g(\frac{k}{n}) + \sum_{k=0}^{\frac{n}{2}-1} g(\frac{k}{n}+\frac{1}{2})\}$$

$$=\frac{1}{n}\{W_n(\frac{n}{2},0)+W_n(\frac{n}{2},\frac{1}{2})\} \qquad \text{for even } n \quad (3.2.a)$$

$$n\ S(n,0) = \sum_{(i<N/2,i\neq 0)} (g_i + g_{N-i})$$

$$= \sum_{(i<N/2,i\neq 0)} g_i + \sum_{(i<N/2,i\neq 0)} g_{N-i} \qquad \text{for odd } n \quad (3.2.b)$$

where $W_n(\frac{n}{2},t)=\sum_{k=1}^{\frac{n}{2}-1} g(t+\frac{k}{n})$, $g_i \equiv g(\frac{i}{N})$, α_i are linear interpolants, and N is the number of samples per unit interval. Notice that we have excluded g_0 from the computational structure for odd n, and this is valid according to Theorem I of chapter 2.

Observation of equations (3.1) and (3.2) shows that the computation of the Riemann sum, S(n,0), can be split into two parts: the first part is concerned with the first half of the samples of g(t); and the second part is concerned with the second half of the samples of g(t). Therefore, we take advantage of this property by performing the AFT on each N/2 samples, summing each two consecutive half period transforms, to yield the AFT per unit interval. In this way, we can compute the Fourier coefficients of a signal of period 1 continuously at the end of each half period, once the registers are full. The computational structure defined by (3.1) and (3.2) can be deduced from

tables 3.2 and 3.6. In architecture I, signal values whose indices add up to N are paired, summed, and placed into the right shift register for subsequent processing. This required an extra N/2-stage shift register and an adder. Since the shift registers can be easily made to operate at 3 or 4 times the speed of the adder, the adder seems to slow down the processing speed of the architecture. In the modified architecture, only two N/2-stage shift registers with parallel load capability are needed to hold the input sequence of length N. The first half of the input sequence is Fourier-transformed, stored, and then accumulated with the Fourier transform of the second half of the input sequence. This is made possible because the elements of each row in tables 3.2 and 3.6 pass through the same computational data path. In this way, the right N/2-stage shift register/adder combination is removed and consequently both speed is improved and silicon area is reduced.

The block diagram of the modified architecture is shown in Fig 3.2. Assume that the signal samples appear at the input sequentially, g_0, g_1, etc. The first N/2 samples are fed, after subtracting g_0 from each sample, to the right N/2-stage shift register when switch S1 is closed (during this time interval S2 is open). While the left N/2-stage shift register is accepting the next N/2 samples (S1 is open and S2 is closed), the outputs of the right shift register are fed through the arithmetic unit to generate the Riemann sums. The results of the arithmetic unit, the values of $\{ S_k \}$, are then transfered to the accumulator unit, where it is added to the previous stored results. During this period of time, there are two operations as well taking place. The first operation is the parallel load transfer of the second N/2 samples from the left shift register to the right shift register to the arithmetic unit to generate the

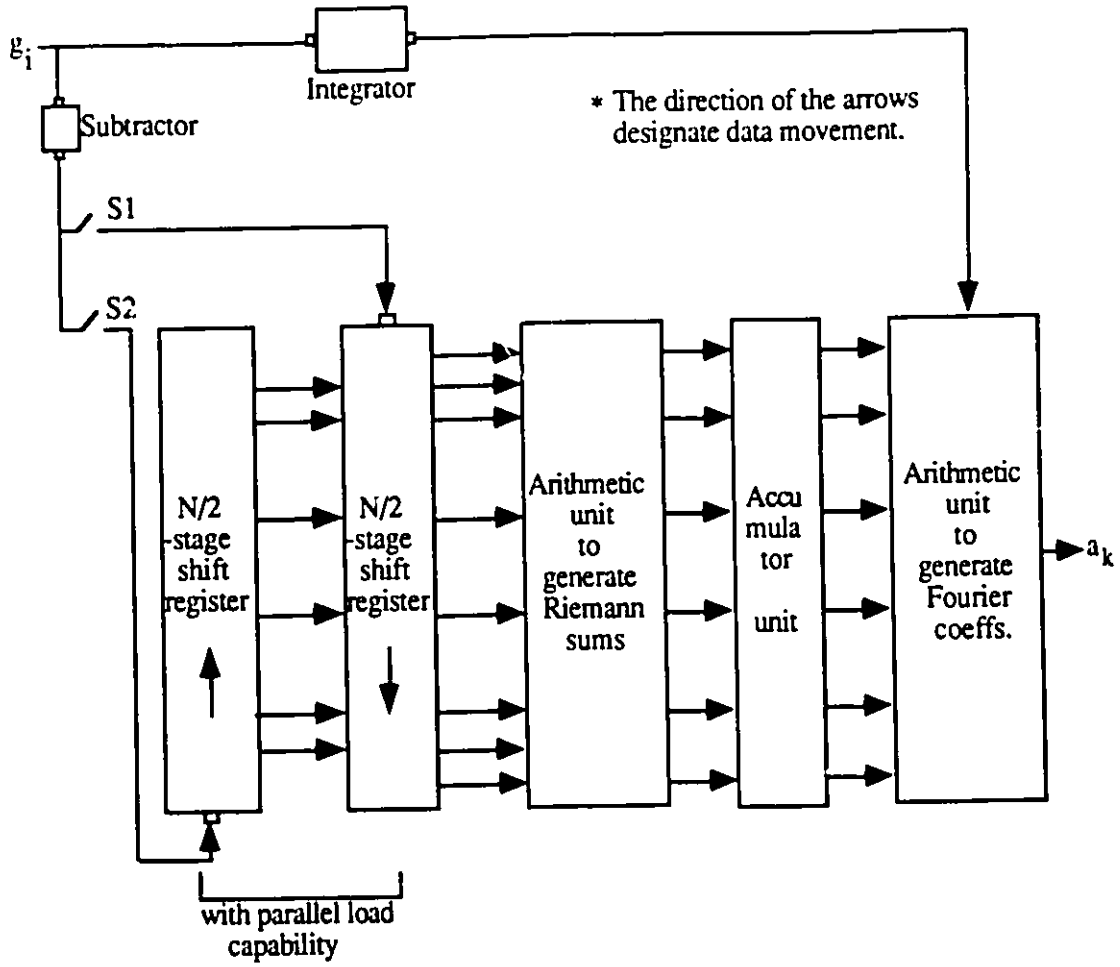


Fig 3.2 : Block diagram of architecture II.

Riemann sums for this portion of the input signal. The second operation is the first half of the samples of the input signal of the next period are fed to the right shift register (of course, after the parallel load transfer sub-operation is executed). The results of the arithmetic unit for the second N/2 samples are then accumulated with the results for the first N/2 samples (stored previously) in the accumulator unit, and fed to the arithmetic unit to generate the AFT of the input signal. The calculation of the mean and the inclusion of the multiplier offset is as discussed in the previous architecture. This cycle repeats itself every one half period of the input signal. The

advantages of this architecture over the previous one can be summarized in to three points:

1- Data transfer and manipulation is eliminated, and thus speed is improved.

2- Number of hardware components is reduced. This leads to smaller silicon area in a VLSI realization.

3- Ability to produce Fourier coefficients at the end of every consecutive half period (i.e., after every N/2 samples of the signal have been processed).

3.4 ARCHITECTURE III

We will consider here the AFT implementation of zero order and linear interpolation procedures for general functions. These functions have both cosine and sine Fourier series expansion, so that both the real and imaginary components of the Fourier transform have to be computed. We will perform the calculations by expressing the AFT in terms of a dot product between a matrix and a vector. We will base the discussion in this section on a 10 sample, 5 coefficient example.

3.4.1 THE AFT MATRIX REPRESENTATION

We shall represent the AFT in vector-matrix notation that is suitable for systolic or processor implementation. Let N be the length of the input sequence and M be the length of the output transform. Let $g_i \equiv g(\frac{i}{N})$ be an input data sequence defined for i=0,1,2,.....,N-1. Then the Riemann sum, S(n,t), of order n is defined by

$$S(n,t) = \frac{1}{n} \sum_{k=0}^{n-1} g(t+\frac{k}{n}) \quad , n=1,2,.....,M. \tag{3.3}$$

where t=0 for computing the real component of the transform and $t=\frac{1}{4k}$ for computing the imaginary component of the transform.

Consider the computation of the real component of the AFT. Equation (3.3) can be written into the matrix form

$$\mathbf{S} = \mathbf{R}\,\mathbf{g} \tag{3.4}$$

where $\mathbf{g} = [\, g_0 , g_1 , g_2 , ..., g_{N-1} \,]$, and R is the MxN matrix whose entries are given by:

Zero-order interpolation:

For i = 1,2,.....,M ; k = 0,1,.....,i-1

$$r_{ij} = \frac{1}{i} \quad , \text{if } j = [\frac{Nk}{i}] \text{ for some k}$$

$$= 0 \quad , \text{otherwise} \tag{3.5}$$

Linear interpolation:

For i = 1,2,.....,M ; k = 0,1,.....,i-1

$$r_{ij} = \frac{i\,(j+1) - N\,k}{i^2} \quad , \text{if } j = [\frac{Nk}{i}] \text{ for some k}$$

$$= \frac{i - (ij - N\,k)}{i^2} \quad , \text{if } j = [\frac{Nk}{i}] + 1 \text{ for some k}$$

$$= 0 \quad , \text{otherwise} \tag{3.6}$$

Consequently, equation (3.4) can be written in the form

Zero order interpolation: For i=1,2,...,M; $$ {}^{i}S_i = \sum_{j=0}^{N-1} r_{ij}' \, g_j \qquad (3.7) $$

Linear interpolation: For i=1,2,...,M; $$ {}^{i^2}S_i = \sum_{j=0}^{N-1} r_{ij}' \, g_j \qquad (3.8) $$

where for zero order interpolation, r_{ij}' ($= i * r_{ij}$) is either 0 or 1, and for linear interpolation, r_{ij}' ($= i^2 * r_{ij}$) is either 0 or real integer. We notice that equations (3.7) and (3.8) are general, and in fact (3.8) is similar to (3.7) for cases where i divides N.

Now define $\mu_{ij} = \mu(ij)$, where μ is the Möbius function. Then the real component of the AFT can be written in the matrix form

$$ a = U S \qquad (3.9) $$

where

S = [S(1,0) , S(2,0) , ...,S(M,0)] is an Mx1 vector

U is an MxM matrix with entries μ_{ij}

$a = [a_1 , a_2 , ... , a_M]$ is an Mx1 vector containing the real component of the AFT

The computation of the imaginary component of the AFT is the same as that of the real component with the exception, of course, t=1/4k and M replaced with M1 in equation (3.4), where M1 is the number of Riemann sums needed

to be calculated (in fact, $M1 = \sum_{i=0}^{q} \sum_{j=1}^{[M/2^i]} | \mu(j) |$; where q is the number of disjoint integer subsets defined by $N_q = \{ 2^q (2m+1) \mid m \in I \}$; q , I =0,1,2,.....).

As an illustration, we consider a 10 sample, 5 coefficient example based on the linear interpolation scheme. Applying equations (3.7) through (3.9), we get

$$\begin{bmatrix} S(1,0) \\ 2S(2,0) \\ 9S(3,0) \\ 8S(4,0) \\ 5S(5,0) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 2 & 1 & 0 & 1 & 2 & 0 & 0 \\ 2 & 0 & 1 & 1 & 0 & 2 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ . \\ . \\ g_9 \end{bmatrix}$$

$$\begin{bmatrix} 2S(1,1/4) \\ 4S(2,1/4) \\ 18S(3,1/4) \\ 10S(5,1/4) \\ 8S(2,1/8) \\ 16S(4,1/8) \\ 32S(4,1/16) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 3 & 3 & 0 & 1 & 5 & 0 & 0 & 5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 3 & 1 & 0 & 0 & 0 & 3 & 1 & 0 & 0 \\ 0 & 3 & 1 & 1 & 3 & 0 & 3 & 1 & 1 & 3 \\ 3 & 5 & 0 & 7 & 1 & 3 & 5 & 0 & 7 & 1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ . \\ . \\ . \\ . \\ g_9 \end{bmatrix}$$

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S(1,0) \\ S(2,0) \\ S(3,0) \\ S(4,0) \\ S(5,0) \end{bmatrix}$$

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} S(1,1/4) \\ S(2,1/4) \\ S(3,1/4) \\ S(5,1/4) \\ S(2,1/8) \\ S(4,1/8) \\ S(4,1/16) \end{bmatrix}$$

Observation of the R matrices reveals the same symmetry encountered in the implementation of architectures I and II. This symmetry can be used to cut the storage requirements by a factor of 2. Note that all of the matrices are sparse, especially the U matrices. In the case of the zero order interpolation procedure, the R matrices will be more sparse than those shown above.

3.4.2 THE ARCHITECTURE

We will consider implementing the zero order and linear interpolation schemes for general functions. The zero order implementation will be a simple subset with the multipliers, r_{ij}', associated with the linear interpolation case reverted to unity multipliers. The single operation common to all of equations (3.7) through (3.9) is the so-called inner product step. This operation is a matrix-vector multiplication, and can be implemented by using systolic arrays [23,24]. However, in this section we will consider a different approach.

Linear Interpolation:

Consider the implementation of the AFT using linear interpolation. Fig 3.3 shows three types of geometries for this implementation. For now consider the computation of the real component of the AFT. In type(a) geometry, the left hand side part of the architecture performs the calculations

associated with equation (3.8) and the right hand side part performs the calculations associated with equation (3.9). The division by n involved in the computation of the Riemann sums is incorporated into the U matrix. The matrix entries r_{ij}', which are pre-defined and pre-stored in a ROM or PROM, are fed to be multiplied row-wise with the input signal, g_j $(0 \le j < N)$. The

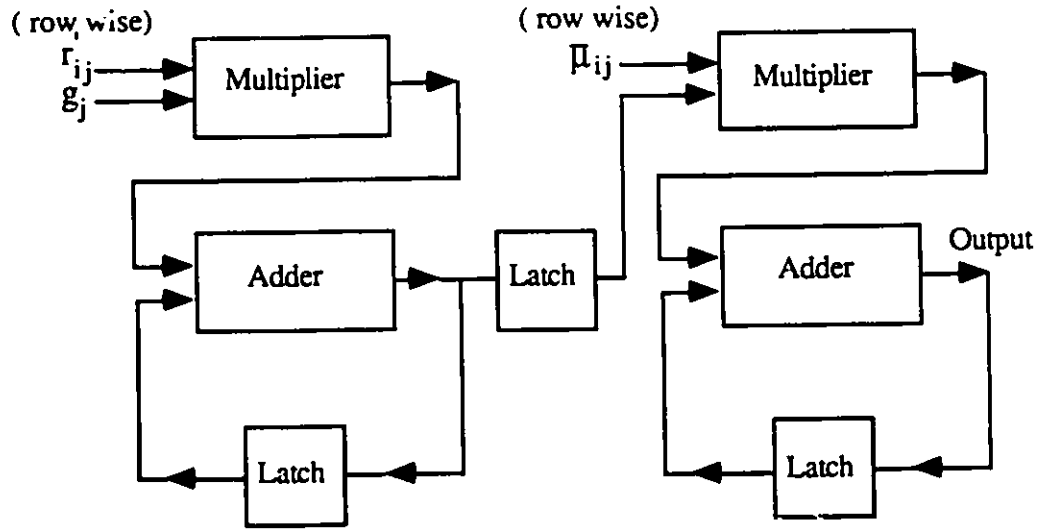


Fig 3.3(a) Type(a) geometry for implementation of the AFT using linear interpolation.

right hand side part of the architecture performs the calculation of (3.9) in a similar way. The pre-defined and pre-stored matrix entries $\bar{\mu}_{ij}$ are fed row-wise, where the bar over μ_{ij} denotes that division by n is built-in. To compute M Fourier coefficients in parallel, the configuration of type(a) geometry is duplicated M times in parallel, with the output of the latch in the middle of each geometry being made accessible to all configurations. Thus each configuration computes the multiplication of one row of (3.7) and (3.8) with a vector. In type(b) geometry, the left hand side part of the architecture is the same as in type(a) geometry. In the right hand side of the architecture, the

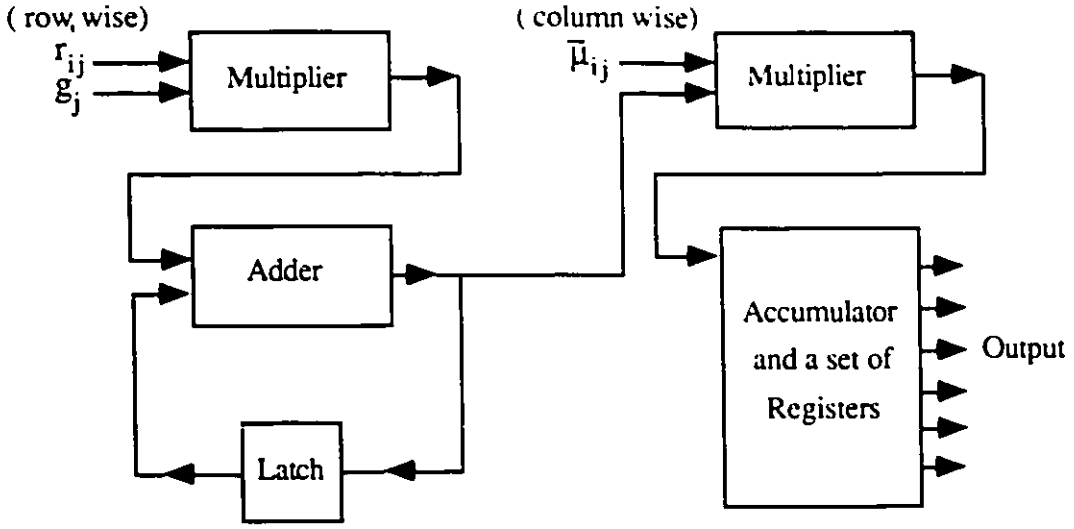


Fig 3.3(b) Type(b) geometry for implementation of the AFT using linear interpolation.

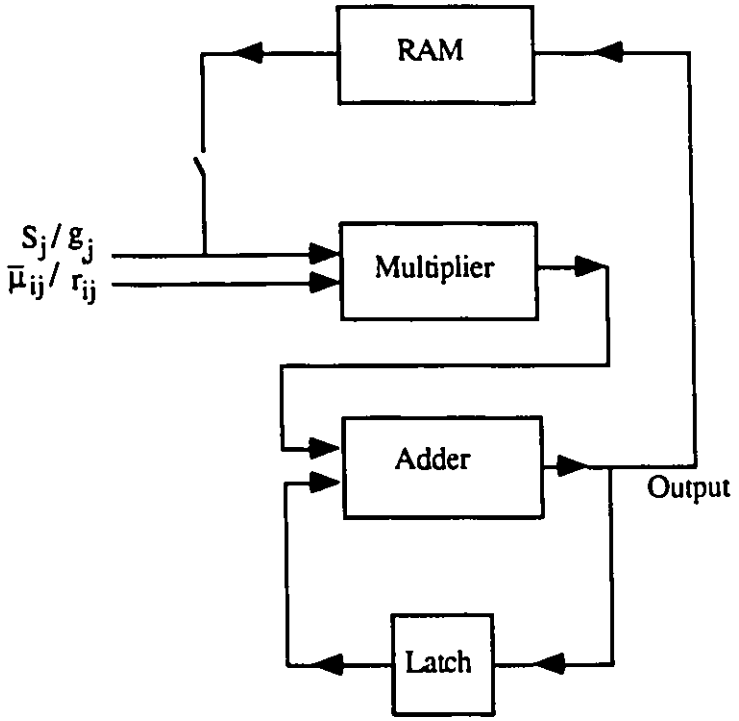


Fig 3.3(c) Type(c) geometry for implementation of the AFT using linear interpolation.

matrix entries $\bar{\mu}_{ij}$ are fed into the multiplier column-wise and the adder/latch combination is replaced by an accumulator and M registers. In

this way, for a non-pipelined configuration, there is an improvement of speed on the order of M^2 units of time over the previous geometry, where a unit of time is defined as the time it takes to perform a single arithmetic operation. Type(c) geometry is a modification of type(a) geometry, where now the intermediate results are stored in a RAM and the same architecture is used for both calculations; calculation of (3.8) is performed first, the results of which are stored in the RAM, followed by calculation of (3.9).

The implementation for computing the imaginary component of the AFT is similar to that of the real component. The three geometries discussed above can be used to compute the imaginary component of the AFT, and therefore, the computation can be performed either in serial (using the same geometry) or in parallel (adding the same geometry in parallel) with the computation of the real component of the AFT.

Zero order interpolation:

Fig 3.4 shows two geometries for zero order interpolation. In both geometries the right hand side part of the architecture operates as before. Since in the zero order interpolation procedure the matrix entries r_{ij}' are either 0 or 1 (in this case the R matrices are Boolean matrices), an accumulator is used to compute the Riemann sums. The input to the accumulator is the input signal, g_j, with r_{ij}' acting as an *Enable signal* . The

Enable signal is pre-defined and pre-stored in a ROM or PROM. If it is a 1, it means that the accumulator should accept the present input and add it to the value of the previous contents. Otherwise, if it is a 0, the accumulator remains idle. Type(a) geometry can be used, in a similar way as in the linear

interpolation case, to compute M Fourier coefficients in parallel. The computation of the imaginary component of the AFT is similar to that of the real component, as discussed previously.

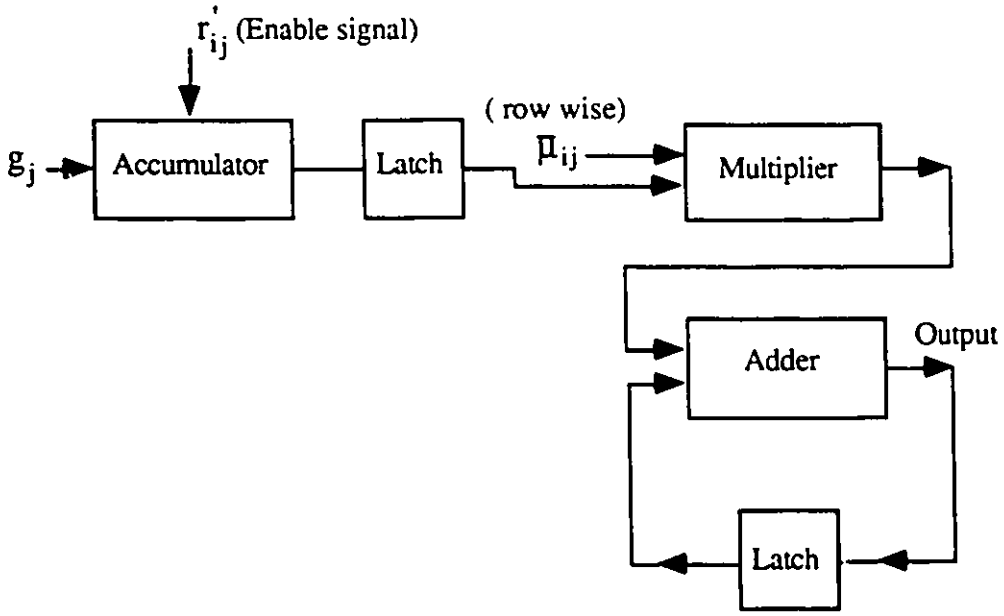


Fig 3.4(a) Type(a) geometry for implementation of the AFT using zero order interpolation.

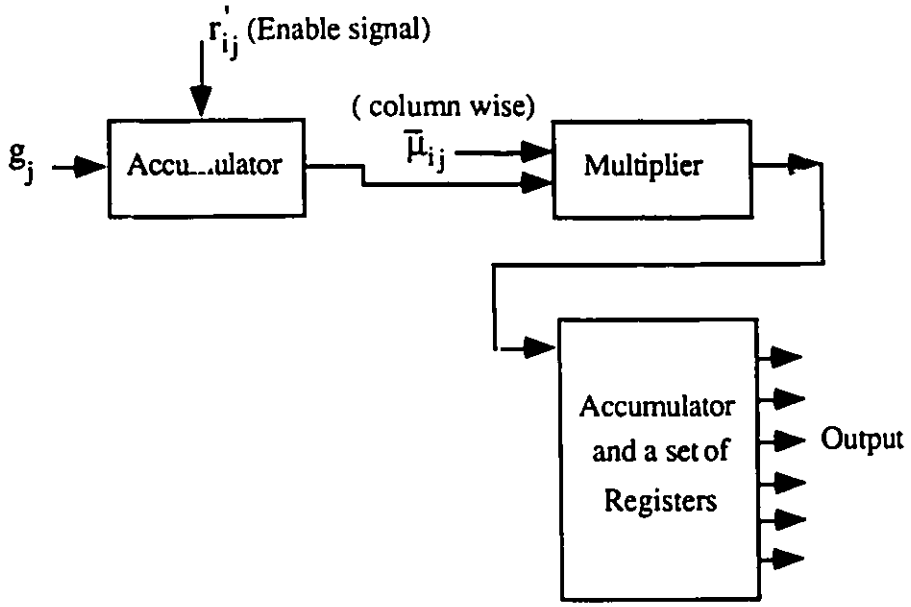


Fig 3.4(b) Type(b) geometry for implementation of the AFT using zero order interpolation.

It is important to note that in all of the geometries presented above, we have acquired idling at the expense of structural regularity and simplicity. The fact that the R and U matrices are sparse leads to idling because of the zero terms present in the computation process. However, in real-time processing, the input samples are sequentially fed to the parallel processors. This will always lead to idling for situations such as the computation of the AFT because of two things. The first is due to the fact that the computation of Fourier coefficients using the AFT does not use all of the samples of the input signal (in contrast, the computation of Fourier coefficients using the DFT or the FFT utilizes all of the samples of the input signal for every Fourier coefficient). This leads to zero terms being present in the R matrices and consequently leads to idling. The second thing is that in real-time processing, the clock rate of the processor should be matched to the clock rate of the input signal for an efficient realization. Therefore, idling is not a problem for real-time processing of the AFT. Only for non real-time processing will idling be a problem and consequently a tradeoff exists between structural regularity and simplicity for computation of the AFT using the inner product approach.

3.4.3 MODIFIED ARCHITECTURE FOR SMALL SIZE AFT

In this section, we present a numerical method for expressing the Arithmetic Fourier transform in terms of a dot product between an MxN matrix of integer entries and a Nx1 input vector, where M is the size of the transform and N is the length of the input sequence. The MxN matrices for calculating the real and imaginary components of transform of a signal have the same highly regular dimensional structure for every M and N, and make the AFT computation possible via systolic architecture. This method

eliminates the high dependence on M and N, and consequently produces a balanced computational scheme for calculating the Fourier coefficients.

Let α_{z_1} and α_{z_2} be the least common multiple of all divisors in equation (3.5), respectively, for computing the real and imaginary components of the transform, and let α_{l_1} and α_{l_2} be the least common multiple of all divisors in equation (3.6), respectively, for computing the real and imaginary components of the transform. Then, equation (3.9) becomes

$$\mathbf{a} = \frac{1}{\alpha_1} U_1 R_1' \mathbf{g} = \frac{1}{\alpha_1} Q_1 \mathbf{g} \qquad (3.10.a)$$

$$\mathbf{b} = \frac{1}{\alpha_2} U_2 R_2' \mathbf{g} = \frac{1}{\alpha_2} Q_2 \mathbf{g} \qquad (3.10.b)$$

where $R_1' = \alpha_1 R_1$, $R_2' = \alpha_2 R_2$, $Q_1 = U_1 R_1'$, $Q_2 = U_2 R_2'$, and where α_1 and α_2 are the appropriate lcm scalars. Equation (3.10) gives the systolic algorithm for computing the Arithmetic Fourier transform. Note that both Q_1 and Q_2 have the same dimensional order (MxN) and both have integer entries. Therefore the systolic algorithm represented by (3.10) produces a balanced computational scheme for calculating the Fourier coefficients, in contrast to the previous algorithms which produced unbalanced computational schemes.

The calculation of the entries of matrices Q_1 and Q_2 can be generated in one step as follows: for $1 \le k \le M$, and $0 \le i \le N-1$,

$$Q_1: \quad q_{ki} = \sum_{j=1}^{[\frac{M}{k}]} \alpha_1 \mu(kj) a_{ji} \qquad (3.11.a)$$

$$Q_2: \quad q_{ki} = \sum_{j=1}^{[\frac{M1}{k}]} \alpha_2 \, \mu(kj) \, a_{ji} \qquad (3.11.b)$$

where M and M1 are as defined previously. We now explain why the algorithm is restricted to small size AFT. The lcm scalars for large size transforms grow very large, especially for linear interpolation schemes, and thus make the implementation impractical using fixed point arithmetic. The integer entries q_{ki} are correspondingly large. However, the systolic algorithm can be used for large size AFT if proper scaling is incorporated to the scheme. On the other hand, for small size transforms, the lcm scalars are relatively small and systolic implementation with integer matrix entries and fixed point arithmetic is possible.

A processor implementation of equation (3.10) is shown in the block diagram of Fig 3.5. A typical Fourier coefficient can be computed by the recurrences

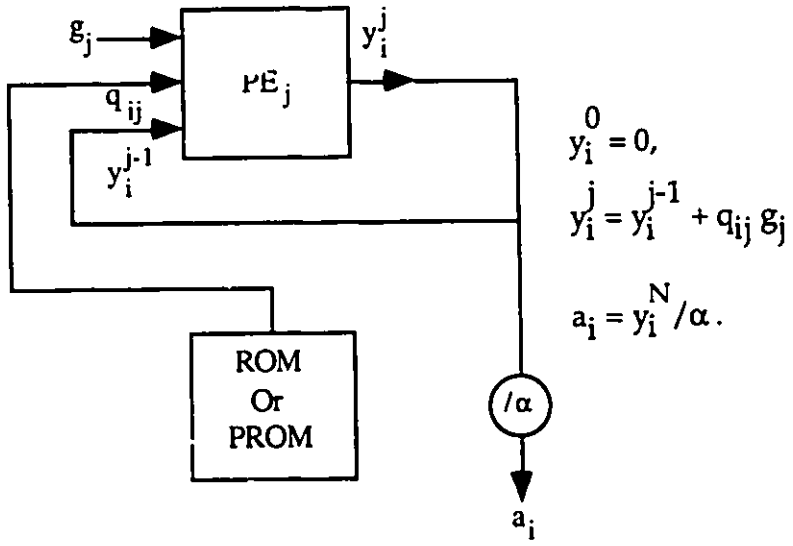


Fig 3.5 : Processor Implementation for small size AFT.

shown beside the figure. The integer matrix entries of Q, q_{ij}, are pre-defined and pre-stored in a ROM or PROM. The processor element, PE_j, executes one multiplication operation and one addition operation every $\frac{1}{N}$ interval of time. Notice that in this architecture the computation of each Fourier coefficient involves one division operation, whereas it is not the case for the previous architectures. A parallel implementation of the AFT for computing M Fourier coefficients using this architecture can be obtained by connecting M such processors in parallel.

3.5 ARCHITECTURE IV

We will consider here the AFT implementation of zero-order interpolation procedure for general functions using memory structures. The motivation for choosing such an architecture is the high modularity and efficient hardware realization that memory-oriented structures offer. It is evident that the butterfly-like structures, that are used in the previous architectures for implementing the AFT, present routing and interconnection problems, in addition to high structural irregularity. The improvements in structural regularity and, perhaps, speed possible with the approach presented in this section does not come from a reduction in the number of arithmetic operations, but by changing the calculation to one of repeated addressing and accumulating. This leads to an efficient VLSI realization of the AFT, consisting of memories, adders, and accumulators with add/subtract capability.

3.5.1 THE ARCHITECTURE

The AFT implementation of zero-order interpolation procedure using memory-oriented architecture is shown in the block diagram of Fig 3.6. The Instruction read-only memory contains pre-calculated look-up tables for signal indices, divisors, and sum indices for a specific AFT size. The signal indices codes address the Signal random-access memory, the divisors codes address the Division ROM, and the sum indices codes address the Sum RAM. The Signal RAM contains the signal values, which are either placed into the accumulator to form the sum or written into it to contain the signal values for the next transform. The Division ROM contains a look-up table for performing the division operation to complete the calculation of the Riemann sum. These Riemann sums are then written into the Sum RAM, and finally are addressed by the Instruction ROM to output the Fourier coefficients through the second accumulator with add/subtract capability. Note that the Fourier coefficients a_k and b_k for k=1,.....,M are computed concurrently.

The sequential operation of the algorithm is explained as follows. The algorithm fetches data from the Signal RAM (where the input signal is stored) according to the addresses in the Instruction ROM, given by the signal indices codes, and places them into the accumulator. For a particular Riemann sum of order n, after n such fetches and accumulations, the Instruction ROM provides the Division ROM with either the appropriate divisor or the address for the appropriate divisor, to complete the computation of the Riemann sum. The result is placed in the Sum RAM. After all of the Riemann sums are calculated and placed in the Sum RAM, the algorithm fetches data from the Sum RAM according to the addresses in

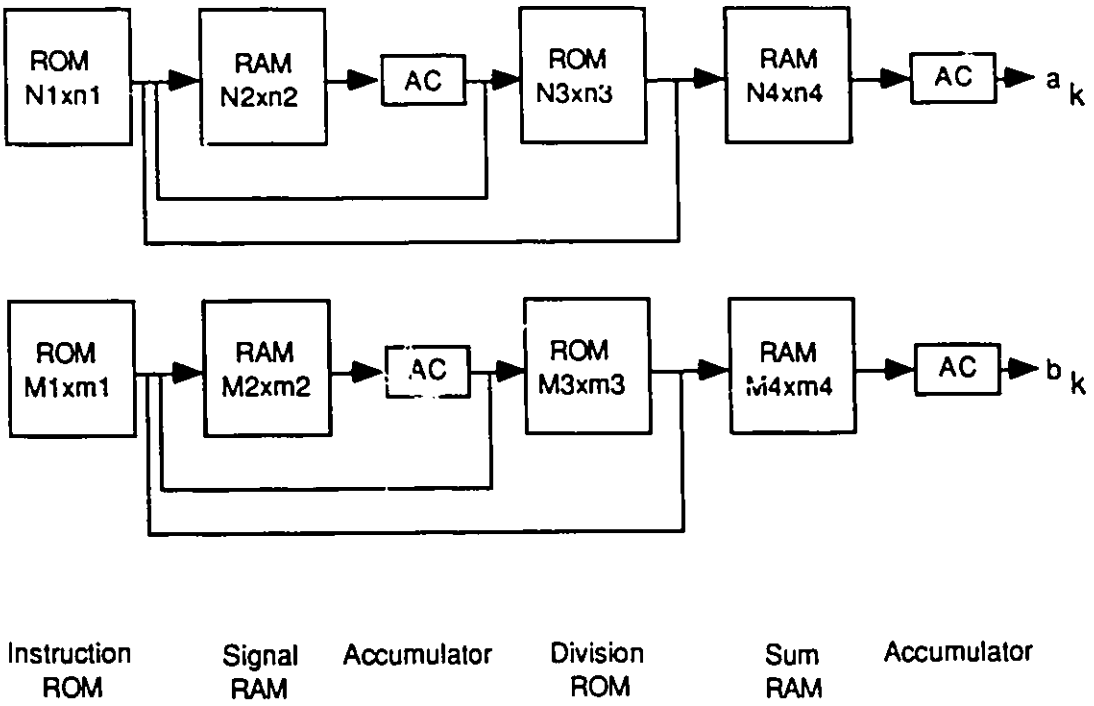


Fig 3.6 : Block diagram of architecture IV.

the Instruction ROM, given by the sum indices codes, and places them into the accumulator with the add/subtract capability. The output from the accumulator after $[\frac{M}{k}]$ cycles yields the Fourier coefficients of the signal, where, k denotes the k-th coefficient, M is the number of Fourier coefficients, and [x] means the greatest integer $\leq x$.

We now propose an implementation for the division ROM. One possible design is shown in Fig 3.7. For direct division, the combined B-bit dividend and n1-bit divisor define a unique address in the ROM. In the diagram, CLB is a combinational logic block which is used to select or address one of (M-1) ROM modules. By including the CLB, we obtain a ROM architecture in which the size of the decoder unit is independent of the divisors. (By not

including the CLB, the resulting ROM architecture will require the $\log_2 M$ bits for the divisors to be added to the input lines of the decoder unit. Consequently, the decoder unit will be large and may be impractical for large set of divisors). Hence, the purpose of the CLB is to alleviate the input loading of the decoder unit.

Each ROM module contains a look-up table (which is addressed by the decoder for a specific divisor) and n3 pass transistors. The contents of the addressed word, out of the 2×2^B possible words in the ROM module, form the n3-bit result (the factor of 2 is included to account for positive as well as negative dividends). At any time, only one out of the (M-1) ROM modules is selected by the CLB. Based on the value of the divisor, the CLB decides which of the ROM modules is to be allowed to pass its output (through the use of the pass transistors) to the next stage. The output of the pass transistors must be reset before each new divisor enters the CLB in order to allow the OR gates to read the correct data. (Notice in the design, we have available at the output of the pass transistor stage the result of division of the operand by all possible divisors).

The bit capacity of the entire ROM must be equal to $(M-1) \times n3 \times 2^{B+1}$ bits. This bit capacity requirement can be reduced as a tradeoff for speed by noting the fact that we can generate divisions by scalars from a certain subset of divisors. This subset of divisors is the set of integers which are prime in the set {2, 3, 4, ..., M}. Thus, the division by a scalar which is not a prime number can be performed in more than one step. The fact that divisions by large scalars which are not prime numbers take more calculation steps than divisions by small scalars does not imply that there will be idling in the computation process. In fact, this is alright since large scalars involve large Riemann sums

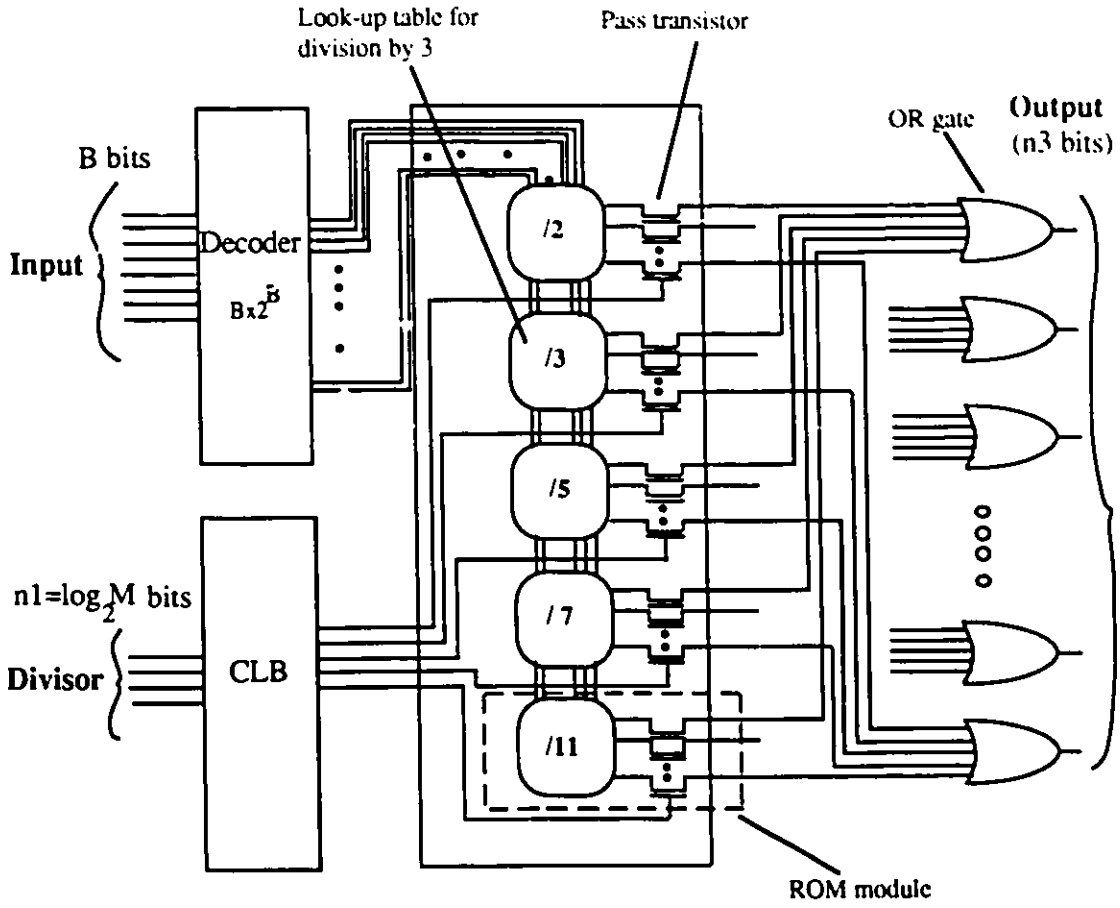


Fig 3.7: Block diagram of the Division ROM.

which will take a longer time to compute.

The bit capacity of the division ROM can be further reduced by converting the implementation of the division operation into one of a multiplication operation [25]. In this case, the combined B-bit multiplicand and n-bit multiplier $(\frac{1}{\text{divisor}})$ define a unique address in the ROM. In [25], it is shown that by partitioning the given numbers as a sum of two numbers, one can reduce the bit capacity requirement by utilizing a ROM-adder multiplication network. A 4k-by-4k ROM-adder multiplication network with output rounding to 4k requires k^2 256-byte ROM modules and k(3k-1)/2 4-bit adders.

Of course, this reduction in ROM area is a tradeoff for speed. The total delay

in such a ROM-adder multiplication network is [25]:

$$\tau = \tau_R + (2k - 1)\,\tau_a \qquad (3.12)$$

where τ_R is memory access time of the ROM and τ_a is the total carry delay in the 4-bit adder.

Another possible design for implementing the division operation would be to replace the Division ROM with a high speed fixed-point divider (e.g., [26-28]). This alternative is straight forward and extensive literature on the design of high speed dividers can be found in ([25], pp. 206-277).

3.5.2 CALCULATING THE MEMORY SIZES

In the following discussion, let N be the length of the input signal (number of input samples per unity interval) and M be the length of the output signal (number of Fourier coefficients). A straightforward implementation of Fig 3.6 requires the following "complex' memory sizes (assuming the signal is complex):

$$N1 = \{\sum_{k=1}^{M}\sum_{j=1}^{[\frac{M}{k}]} |\,\mu(j)\,| \sum_{r=0}^{jk-1} 1\} + 2M = \{\sum_{k=1}^{M}\sum_{j=1}^{[\frac{M}{k}]} jk\,|\,\mu(j)\,|\} + 2M$$

$$N2 = N$$

$$N3 = 2\,(M-1)\,2^{B} = (M-1)\,2^{B+1}$$

(The factor of 2 is used to account for positive and negative inputs)

$$N4 = M$$

$$M1 = \{\sum_{k=1}^{M}\sum_{j=1}^{[\frac{M}{k}]} jk\,|\,\mu(j)\,|\} + 2\sum_{i=0}^{q}\sum_{j=1}^{[M/2^{i}]} |\,\mu(j)\,|$$

$$M2 = N$$

$$M3 = (M-1)\,2^{B+1}$$

$$M4 = \sum_{i=0}^{q} \sum_{j=1}^{[M/2^i]} | \mu(j) |$$

where q is defined by $N_q = \{ 2^q (2m+1) \mid m \in I \}$, q, I = 0,1,2,..... . Given N1 and N2, which are functions of the transform size, and given the word size, B, for the input data representation, the following constraints must hold (assuming that the word size is kept at B bits during the entire computation scheme, i.e., output rounding or truncation is applied at each stage of the computing process) :

For a_k :

$N1 = \text{given}$

$N2 = N$

$N4 = M$

$n2 = n3 = n4 = B$

$n1 = \text{Max} \{ \log_2 N, \log_2 M \} = \log_2 N$

$N3 = (M-1)\,2^{B+1}$

The same constraints apply for computing the b_k's, except for m1 since the computational load for computing the b_k's is greater. This results in an increase of m1 by 1 or 2 bits.

Consider as an example the 60 sample, 12-coefficient system used earlier. The choice for the value of B is, of course, dependent on the required accuracy and consequently the signal processing system in which the AFT to be used. We assume that the number of bits (B) representing the input signal is 8 bits.

According to Equations (4.26) and (4.27), the upper and lower bounds for the root-mean-square of the error due to finite wordlength computation will be, respectively, 3.22×10^{-2} and 5.52×10^{-3}. However, it turns out, as we will see in chapter 4, that a realistic and a reasonable error bound ($\approx 10^{-5}$) will require the input signal to be represented by 14 bits. The memory sizes for the architecture are calculated as follows (assuming signal is real) :

For a_k (k=1,2,.....,12):

signal indices = 78

divisors = 11

sum indices = 12

Therefore, N1 = 101 words

n2 = n3 = n4 = 8 bits

N2 = 60 words

N4 = 12 words

$n1 = \log_2 N = 6$ bits

$N3 = 11 \times 2^9 = 5.632$ K words (direct)

$= 5 \times 2^9 = 2.56$ K words (prime decomposition)

For b_k (k=1,2,.....,12):

signal indices = 120

divisors = 18

sum indices = 18

Therefore, M1=156 words

m2 = m3 = m4 = 8 bits

M2 = 60 words

M4 = 18 words

m1 = 7 bits

M3 = 5.632 K words (direct)

= 2.56 K words (prime decomposition)

Note that the ROM size (N3 and M3) for this example is reduced by a factor of 2 when prime decomposition is utilized as shown in Fig 3.7. The tradeoff for this saving in hardware will be a reduction in speed by a factor of 3. Note also that we can share the division ROM for computation of both the real and imaginary components of the AFT and consequently only 20.5 K bits of ROM (8 bits for the dividend, 6 bits for the divisor, and 8 bits for the result) is required for implementing the division for the 12 coefficient, 60 sample example. If we implement the divisions as multiplications, then the hardware requirements of the multiplication network [25] of size 8 x 8 with output rounding is 5 4-bit adders and 8.192 K bits of ROM. This hardware requirement is independent of the size of the AFT insofar as the size of the operands are kept constant. In general, as the size of the output transform becomes large, the accuracy requirement causes the size of the multiplier to be increased, and consequently the hardware requirements for the multiplication network increases.

3.6 ARCHITECTURE V

We will consider here the AFT implementation of zero-order interpolation procedure for general functions. We will perform the computation of the AFT by considering special patterns and symmetries associated with computation of the averages, S(n,t). The discussion in this section will be based on the 60 sample, 12 coefficient example used earlier.

S_1	g_0
S_2	$\frac{1}{2}(g_0 + g_{30})$
S_3	$\frac{1}{3}(g_0 + g_{20} + g_{40})$
S_4	$\frac{1}{4}(g_0 + g_{15} + g_{30} + g_{45})$
S_5	$\frac{1}{5}(g_0 + g_{12} + g_{24} + g_{36} + g_{48})$
S_6	$\frac{1}{6}(g_0 + g_{10} + g_{20} + g_{30} + g_{40} + g_{50})$
S_7	$\frac{1}{7}(g_0 + g_{17} + g_{34} + g_{51}) + \frac{1}{7}(g_9 + g_{26} + g_{43})$
S_8	$\frac{1}{8}(g_0 + g_{15} + g_{30} + g_{45}) + \frac{1}{8}(g_7 + g_{22} + g_{37} + g_{52})$
S_9	$\frac{1}{9}(g_0 + g_{20} + g_{40}) + \frac{1}{9}(g_7 + g_{27} + g_{47}) + \frac{1}{9}(g_{13} + g_{33} + g_{53})$
S_{10}	$\frac{1}{10}(g_0 + g_6 + g_{12} + g_{18} + g_{24} + g_{30} + g_{36} + g_{42} + g_{48} + g_{54})$
S_{11}	$\frac{1}{11}(g_0 + g_{11} + g_{22} + g_{33} + g_{44} + g_{55}) + \frac{1}{11}(g_5 + g_{16} + g_{27} + g_{38} + g_{49})$
S_{12}	$\frac{1}{12}(g_0 + g_5 + g_{10} + g_{15} + g_{20} + g_{25} + g_{30} + g_{35} + g_{40} + g_{45} + g_{50} + g_{55})$

Table 3.8 Procedure for S values

First, let us consider the implementation for computing S_k, $1 \le k \le 12$. There is a pattern associated with computation of the averages. We observe from table 2.3 (chapter 2, section 5) that for averages whose index, k, divides N the summation terms of S_k are spaced in the time domain by the value $\frac{N}{k}$. The rest of the averages whose index does not divide N, the summation terms of S_k can be split into two sub-summations: the first sub-summation consists of terms which are spaced in the time domain by the value $[\frac{2N}{k}]$; and the second sub-summation consists of a delayed version of the first sub-summation, with the delay given by the value $[\frac{N}{k}] + 1$, where [x] denotes the greatest integer $\le x$. For example, S_4 belongs to the case for which k divides N, since 4

$B_{1\,1}$	g_{15}
$B_{1\,2}$	$\frac{1}{2}(g_{15} + g_{45})$
$B_{1\,3}$	$\frac{1}{3}(g_{15} + g_{35} + g_{55})$
$B_{1\,5}$	$\frac{1}{5}(g_3 + g_{15} + g_{27} + g_{39} + g_{51})$
$B_{1\,6}$	$\frac{1}{6}(g_5 + g_{15} + g_{25} + g_{35} + g_{45} + g_{55})$
$B_{1\,7}$	$\frac{1}{7}(g_7 + g_{24} + g_{41} + g_{58}) + \frac{1}{7}(g_{15} + g_{32} + g_{49})$
$B_{1\,9}$	$\frac{1}{9}(g_{\_} + g_{22} + g_{42}) + \frac{1}{9}(g_8 + g_{28} + g_{48}) + \frac{1}{9}(g_{15} + g_{35} + g_{55})$
$B_{1\,10}$	$\frac{1}{10}(g_3 + g_{15} + g_{27} + g_{39} + g_{51}) + \frac{1}{10}(g_9 + g_{21} + g_{33} + g_{45} + g_{57})$
$B_{1\,11}$	$\frac{1}{11}(g_4 + g_{15} + g_{26} + g_{37} + g_{48} + g_{59}) + \frac{1}{11}(g_9 + g_{20} + g_{31} + g_{42} + g_{53})$
$B_{2\,2}$	$\frac{1}{2}(g_7 + g_{37})$
$B_{2\,4}$	$\frac{1}{4}(g_7 + g_{37}) + \frac{1}{4}(g_{22} + g_{52})$
$B_{2\,6}$	$\frac{1}{6}(g_7 + g_{17} + g_{27} + g_{37} + g_{47} + g_{57})$
$B_{2\,10}$	$\frac{1}{10}(g_2 + g_{14} + g_{26} + g_{38} + g_{50}) + \frac{1}{10}(g_8 + g_{20} + g_{32} + g_{44} + g_{56})$
$B_{2\,12}$	$\frac{1}{12}(g_2 + g_{12} + g_{22} + g_{32} + g_{42} + g_{52}) + \frac{1}{12}(g_7 + g_{17} + g_{27} + g_{37} + g_{47} + g_{57})$
$B_{4\,4}$	$\frac{1}{4}(g_4 + g_{34}) + \frac{1}{4}(g_{19} + g_{49})$
$B_{4\,8}$	$\frac{1}{8}(g_4 + g_{19} + g_{34} + g_{49}) + \frac{1}{8}(g_{11} + g_{26} + g_{41} + g_{56})$
$B_{4\,12}$	$\frac{1}{12}(g_4 + g_{14} + g_{24} + g_{34} + g_{44} + g_{54}) + \frac{1}{12}(g_9 + g_{19} + g_{29} + g_{39} + g_{49} + g_{59})$
$B_{8\,8}$	$\frac{1}{8}(g_2 + g_{17} + g_{32} + g_{47}) + \frac{1}{8}(g_9 + g_{24} + g_{39} + g_{54})$

Table 3.9 Procedure for B values

divides 60. The summation terms of S_4 , therefore, are spaced in the time domain by the value $\frac{60}{4} = 15$, as evident from table 2.3. As another example, consider the computation of S_7 which clearly belongs to the case for which k does not divide N. In this case we form the first sub-summation consisting of terms spaced in the time domain by the value $[\frac{2\times60}{7}] = 17$, i.e., $S_7 = \frac{1}{7}(g_0 + g_{17} + g_{34} + g_{51})$. The second sub-summation consists of terms spaced by the same

value, 17, but with a delay equal to $[\frac{60}{7}]+1=9$, i.e., $S_7 = \frac{1}{7}(g_9 + g_{26} + g_{43})$.

Table 3.8 depicts the procedure for computing S_k, $1 \le k \le 12$. For implementing the computation of S(mk,1/4k) in table 2.4, we apply the same procedure as outlined above, with the argument k replaced by mk. In all cases, there is a delay associated with them. The delays can not be casted into a general exact formula. The procedure for computing $B_{k\,mk}$, $1 \le k \le 12$, $1 \le m \le 12$, $1 \le mk \le 12$, is depicted in table 3.9.

3.6.1 THE ARCHITECTURE

The AFT implementation of zero-order interpolation procedure using the approach outlined above is shown in the block diagram of Fig 3.7. In the diagram the controlled switches are used to control the flow of samples to be placed into the accumulator. In a hardware implementation, this can be realized by using a demultiplexer or by using a complex clocking scheme. The block on the right hand side of the architecture consists of a set of adders and subtractors, which are needed to form the linear combinations of the Riemann sums to compute the Fourier coefficients.

The sequence of operation of the architecture is as follows. Assume that the signal samples appear at the input sequentially, g_0, g_1, etc. After subtraction of g_0 from each sample, the particular accumulator collects the appropriate samples through the controlled switch. The result in the accumulator is then passed to the divider, where the division by the divisors of table 3.7 takes place, to form the Riemann sum. Once the Riemann sums are computed they are fed to the final block of the architecture where they are summed accordingly to generate the Fourier coefficients of the input signal. The role of

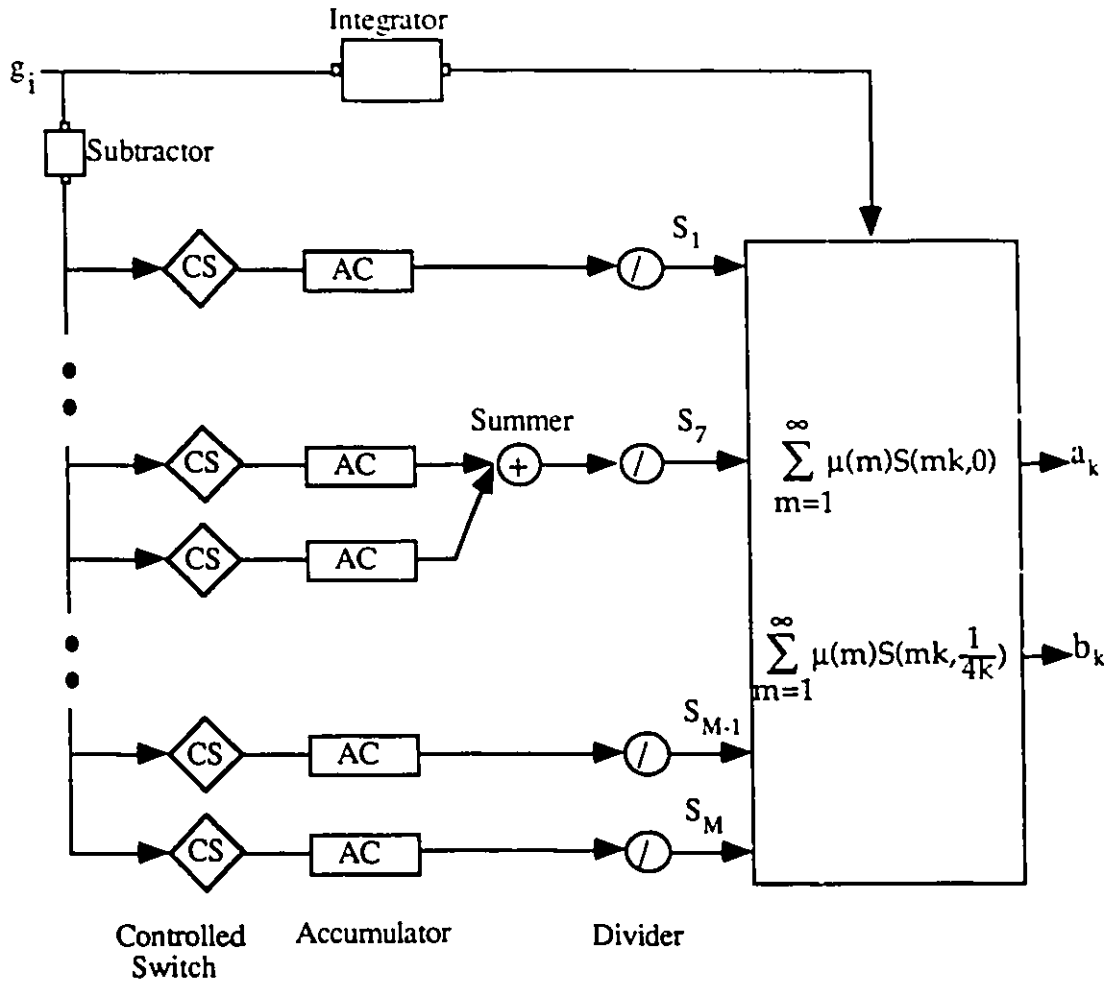


Fig 3.8 : Block diagram of architecture V.

the integrator is, as before, to find the mean of the signal. The inclusion of the offset multipliers, table 3.4, is delayed until the final summation for the Fourier coefficients.

3.7 SUMMARY

In this chapter we have considered implementation schemes for realizing the AFT algorithm. The AFT implementations were based on zero order and linear interpolation procedures. In all the discussion, the aim was to exploit certain symmetry properties of the AFT and to present the basic

principles of efficient computation of the AFT algorithm. Our approach was to illustrate the basic principles with examples.

We have proposed several new architectures for VLSI implementation of the AFT algorithm. Two of the proposed architectures are suitable for computing the Fourier coefficients of symmetric signal using both zero order and linear interpolation procedures. The third architecture includes several different geometries suitable for computing the Fourier coefficients of any complex-valued signal using both zero order and linear interpolation procedures. The final two architectures are suitable for implementation to compute the AFT for any complex-valued signal using zero order interpolation procedure. The proposed architectures are shown to offer an alternative approach to Fourier analysis and synthesis.

CHAPTER 4

SIMULATIONS OF THE PROPOSED ARCHITECTURES USING EXTEND

4.1 INTRODUCTION

The present chapter contains Extend simulations of the proposed architectures for the implementation of the AFT algorithm. The discussion is based on the 12 coefficient, 60 sample example used earlier, which should be general enough to verify the correctness of these architectures. Also in this chapter, discussion on dynamic range growth, quantization noise due to finite wordlength computation, and critical path estimate for each architecture is included. As a final contribution, a comparison between the proposed architectures and between these architectures and the FFT is provided.

4.2 ABOUT EXTEND

Extend is a modeling tool which can be used to develop behavioral models of real-life complex processes. These models enable the user to simulate the actual behavior by creating a block diagram of the process. Because the definition of each block can be quite simple, Extend provides a

powerful tool and the means to test very large complex processes by breaking them into small, simple processes that are easier to handle and test.

Basically, simulating with Extend consists of creating block definitions that make up the system or model. The definitions of these blocks are written in a language, similar to C, called ModL. The language structure or ModL script is made up of a series of message handlers which deal with the messages sent by the user (user messages) and Extend (system messages).

The process of simulating the proposed architectures with Extend consists of the following simple steps:

–creating the appropriate block definitions (e.g., shift register, multiplier, latch, clocks, control units, signal generator, etc)

–building libraries of frequently used blocks

–connecting these blocks to form the architecture or system to be simulated

–running the simulation

Extend supports a plotter utility for displaying the output of the simulation as well as a tabulated output data. More information on Extend is available in the manual.

4.3 SIMULATION OF ARCHITECTURE I

We will consider implementing the linear interpolation scheme for computing 12 Fourier coefficients from 60 samples. The zero order implementation will be a simple subset with both summed sample pairs and multiplier reduction. The simulation block diagram for hardware implementation using architecture I [18] is shown in Fig 4.1. The results of the simulation, i.e., the output at each time step of the simulation, is shown in

Appendix E. We used the sequence $\{ 1 - 30 (\frac{k}{60}) + 60 (\frac{k}{60})^2 - 30 (\frac{k}{60})^4 ; k=0,1,....,59 \}$

as the input signal. This is a modified version of the polynomial test in [18]. It is important to note that the arithmetic elements are not pipelined in this simulation; they compute when all samples are correctly positioned in the right hand 30-stage register. In a silicon realization, we will pipeline the arithmetic elements along with the formation of the sample sums in this register. In this way we can probably reduce the hardware requirements, either by sharing high speed common hardware (e.g. multipliers) or by building very simple low speed hardware for each element.

The total hardware requirement for this example is as follows:

1	Multiplier/Accumulator with divider. This is used to obtain the mean value of the input sequence
2	30 stage shift registers
1	30 stage parallel load register
21	multipliers (maximum multiplicand is 10)
11	dividers
58	2-input adders
8	2-input subtractors

The polynomial generator is used as a test input to the system, and the outputs are then verified by a plotter utility (not shown). The parallel load shift register is used to reverse the first half of the samples where they are summed with the latter half of the samples using the alignment shown in Table 3.2. A VLSI realization of a 4-stage-8-bit parallel load shift register is presented in Appendix F. The output summed pairs are fed through a set of

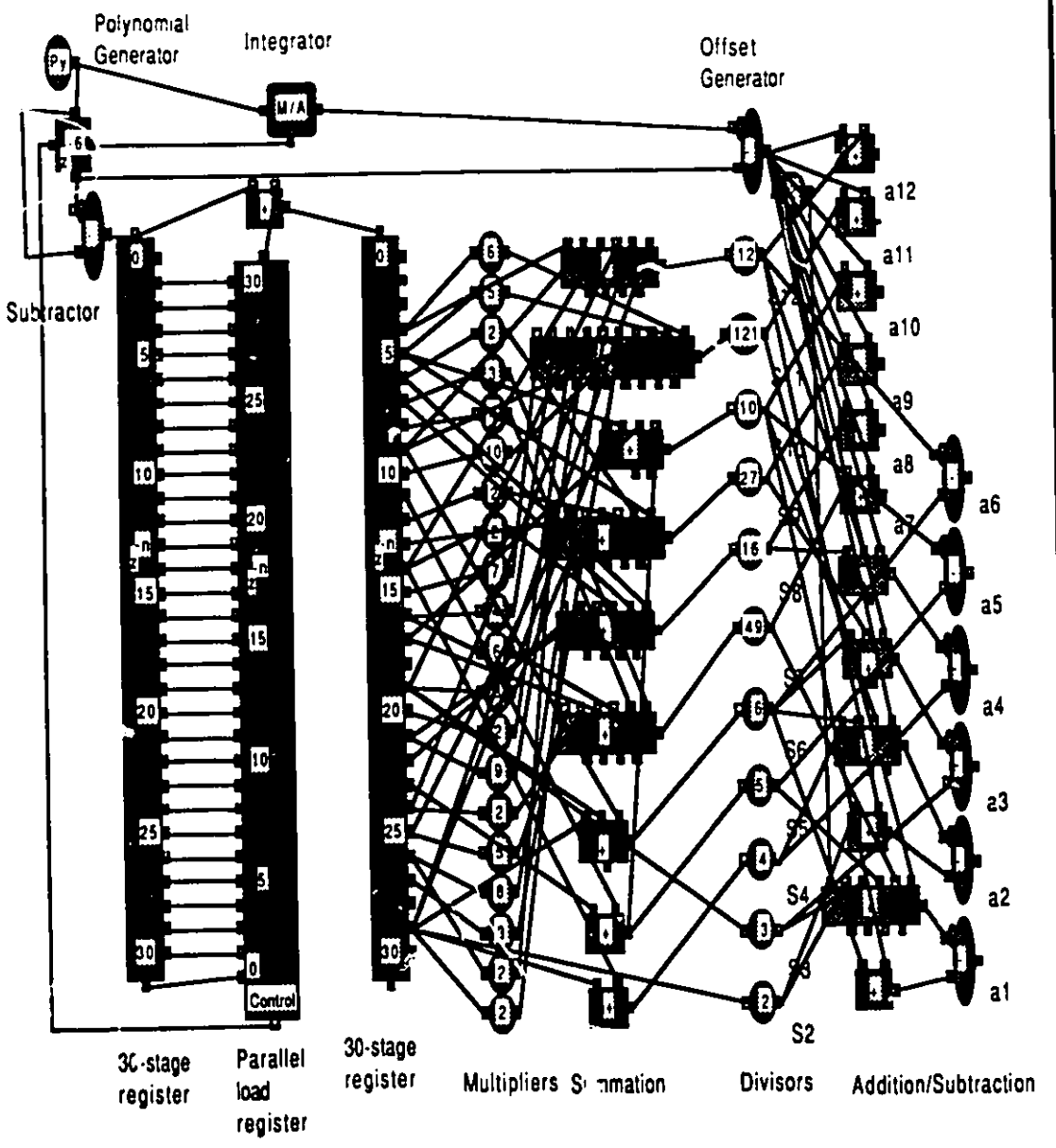


Fig 4.1 : Hardware implementation of architecture I for a 12 coefficient, 60 sample system using linear interpolation.

multipliers and summed (according to Table 3.2) and a division performed with the divisors of Table 3.3. The resulting values of $\{S_i\}$ are summed accordingly. The offset, $(g_0 - \bar{g})$ is generated by the offset generator, and clearly is incorporated to the computation at a stage where the mean value of the

input signal will be able to have been computed. The input subtraction of g_0 from each sample is performed through the subtractor in the register input. The zero order case is a subset of this structure with no multipliers and only a set of divisors. The divisors are small , < 13 for the zero order case.

4.3.1 DYNAMIC RANGE GROWTH

Let N and M be the length of the input sequence and the output sequence, respectively. The maximum multiplier is M-2 (assuming M-1 to be a prime number) and therefore, the maximum dynamic range growth (DRG) is $\log_2(M-2)$. If a tree structure is employed for calculation of the Riemann sum, as shown in Fig 4.2, the maximum number of cascades of 2-input adders is $\log_2(M-2)$ (not including the summation of the input pairs). The output summation stage has maximum DRG of $\log_2 M$ (assuming divisions do not contribute to DRG). The combination is a worst case dynamic range growth of

$$DRG \approx 2\log_2(M-2) + \log_2 M + 1 \qquad (4.1)$$

This is an upper bound on DRG, and in fact exactly true for the computation of the first component of the AFT. The DRG is proportional to $\log_2 M$, where M is the output transform size. The dependence of DRG on N, the length of the input sequence, is implicit since as N approaches the Farey sampling rate the multipliers revert to unity multipliers, which in turn will affect the above equation.

For the 12 coefficient, 60 sample example, the maximum DRG is 11 bits. Thus it is clear that all of the multiplications and summations can be performed with zero error using only a modest computational dynamic range. The division operations will, in general, produce errors due to finite wordlength

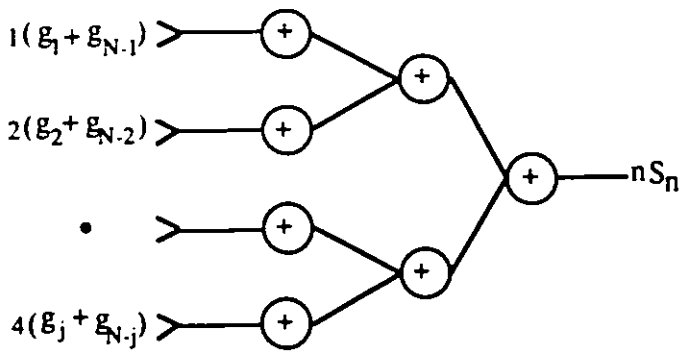


Fig 4.2 : Tree structure for calculation of a Riemann sum.

effects, but this is the only part of the algorithm where noise injection occurs if dynamic range growth considerations are properly incorporated into the architecture.

For the zero order interpolation case, the maximum DRG is given by the following formula:

$$DRG \approx \log_2(M/2) + \log_2 M + 1 \qquad (4.2)$$

The dynamic range growth is solely due to addition operations. For the example above, the maximum DRG is 7 bits, a reduction of 4 bits from the linear case, due to elimination of multiplication operations.

4.3.2 QUANTIZATION ANALYSIS

In this section, we consider the fixed-point accuracy of implementing the AFT using architecture I. The theory and notation for the finite wordlength effects is discussed in Appendix D. The following analysis leads to an upper and lower bound on the root-mean-square error. The approach is similar to one used previously [22,29-32], to predict output noise variance in digital filters and FFT algorithms.

In what follows, we will assume that the numbers are scaled so that the

binary point lies at the extreme left. We will also assume that the input samples (i.e., the real and imaginary parts of g_k) are each represented by B bits plus a sign. The statistical model for the noise assumes the following: (1) the error is a white-noise process; (2) the probability distribution of the error is uniform over the range of quantization error; (3) the error is uncorrelated with the input signal, i.e. the error is independent of the signal, and with other errors. These assumptions appear to be valid if the signal is sufficiently complex and the quantization steps sufficiently small so that the amplitude of the signal is likely to traverse many quantization steps in going from sample to sample. This statistical model only works well for multiplication. For addition, the error is correlated with the input signal and consequently the statistical model does not apply. However, for addition the correlated assumption can be relaxed if we assume that half of the input values are positive and half are negative.

The AFT algorithm for symmetric signal is defined by the equation

$$a_k = \sum_{m=1}^{[M/k]} \mu(m)\, S(mk,0) \quad ; k=1,2,.....,M \tag{4.3}$$

where $S(n,0) = \frac{1}{n} \sum_{k=0}^{n-1} g(\frac{k}{n})$ is the Riemann sum of order n and μ is the Möbius function, which can take one of three possible values: 1, 0, and -1.

We note that Equation (4.3) involves the largest amount of computation when k = 1. Therefore, we will specifically determine the error bound for k=1, as it amounts to the largest error possible in the computation of the AFT. Thus the variance of a_1 is

$$Var(a_1) < \sum_{n=1}^{M} Var(S_n) \tag{4.4}$$

where we have used the notation $S_n = S(n,0)$. The inequality in (4.4) holds because μ can take the value zero and consequently the corresponding product is not needed in computing a_1. We assumed in (4.4) that in the last stage of the algorithm, the computation of Fourier coefficients from linear combinations of Riemann sums can be performed without error. This assumption is valid since the linear combinations are merely additions and subtractions, and these can be performed with zero or small error.

First, we will consider the implementation of the AFT using linear interpolation. The implementation of the AFT using zero order interpolation is a simple subset, with multiplication operations eliminated. Because in the linear interpolation procedure the number of samples and the divisor involved in computing the Riemann sum are not the order, n, but depends on whether n divides the length of the input sequence, N, an accurate analysis of the quantization noise will be quite involved and complicated. Generally, a precise analysis of quantization error is not required in practical applications. For example, a common objective of error analysis is to choose the register length necessary to meet some specifications on the relative sizes of signal and errors. Therefore, we will make the following simplifications in order to derive an expression for the ratio of the rms of the error to the rms of the output transform:

(1) The number of samples, N, is large enough such that all of the integers 1,2,.....,M/2 divide N. This ensures that at least the first half of $\{ S_n \}$ do not involve errors due to linear interpolation approximation.

(2) All of the latter half of $\{ S_n \}$ are affected by linear interpolation of the worst scenario. That is,

$$S(n,0) = \frac{1}{n^2} \sum_{k=1}^{n-1} k \left(g_{[\frac{Nk}{n}]} + g_{N-[\frac{Nk}{n}]} \right) \quad \text{for } n > \frac{M}{2} \qquad (4.5)$$

where the square brackets are subscripts of g. Notice that we assumed that (4.5) applies for all $n > \frac{M}{2}$, which is a conservative estimate since (4.5) applies only when n is a prime number. Now, since we assumed $n > \frac{M}{2}$ to be prime numbers, then the various multipliers, k, within the summation are bounded by M-1, the largest prime in the coefficient index set.

The flow graph for a typical calculation in the AFT implementation using architecture i is shown in Fig 4.3(a). Figure 4.3(b) shows the same structure with noise sources added to account for the effects of rounding the products $k \left(g_{[\frac{Nk}{n}]} + g_{N-[\frac{Nk}{n}]} \right)$, overflow as a result of addition of the sample values, and truncation due to the division operation. We assume the input signal to be complex. In the figure, the complex quantities ε_m, ε_a, and ε_d, represent, respectively, the errors due to rounding, overflow, and truncation. The rounded complex product can be represented as

$$k \left(g_{[\frac{Nk}{n}]} + g_{N-[\frac{Nk}{n}]} \right) = k \, \mathrm{Re}\left[\left(g_{[\frac{Nk}{n}]} + g_{N-[\frac{Nk}{n}]} \right) \right] + \varepsilon_{m1}$$

$$+ k \, \mathrm{Im}\left[\left(g_{[\frac{Nk}{n}]} + g_{N-[\frac{Nk}{n}]} \right) \right] + \varepsilon_{m2} \qquad (4.6)$$

That is, each real multiplication contributes a roundoff error. To compute the variance of the error in S_n, we assume that the errors are uncorrelated with one another and with the input signal. The mean of the error due to

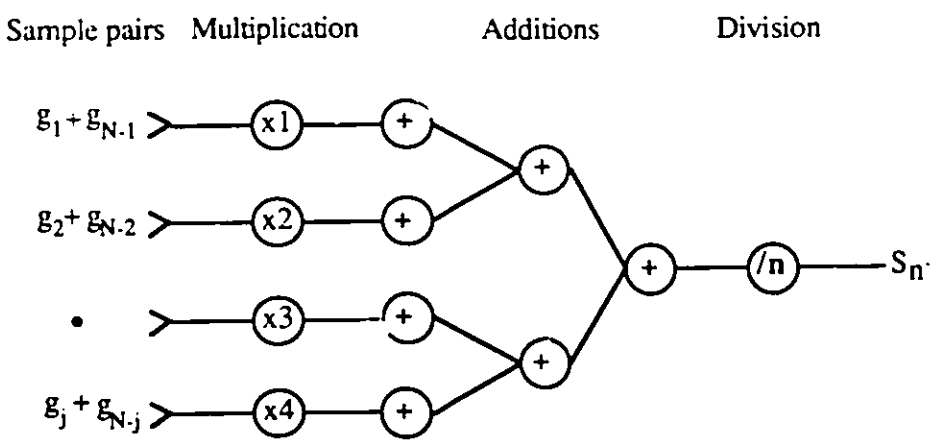


Fig 4.3(a) : Flow graph for architectures I.

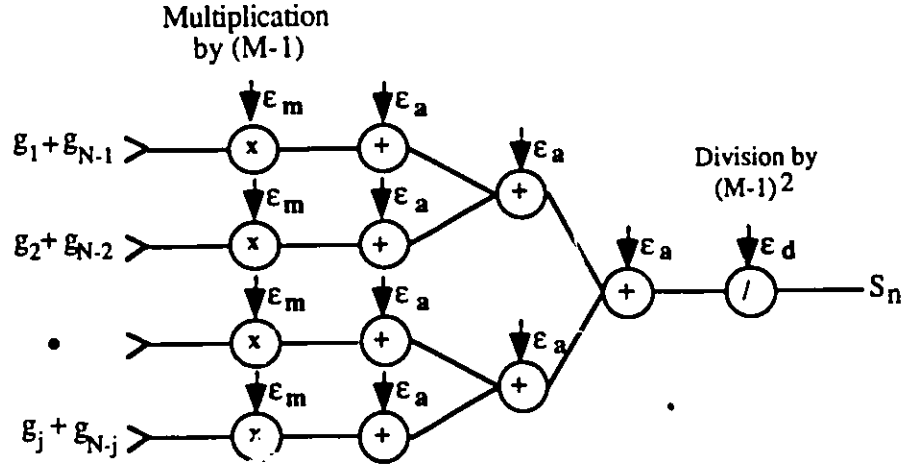


Fig 4.3(b) : Flow graph with noise added for architectures I.

rounding a complex multiplication is zero. Since the squared magnitude of the complex error ε_m is

$$|\varepsilon_m|^2 = \varepsilon_{m1}^2 + \varepsilon_{m2}^2$$

the average value of $|\varepsilon_m|^2$ is

$$E[\,|\varepsilon_m|^2\,] = 2\,\sigma_m^2 = 2\frac{2^{-2B}}{12} = \frac{2^{-2B}}{6}$$

And similarly, the average values of $|\varepsilon_a|^2$ and $|\varepsilon_d|^2$ are

$$E[\,|\varepsilon_a|^2\,] = 2\,\sigma_a^2 = 2\,\frac{2^{-2B}}{2} = 2^{-2B}$$

$$E[\,|\varepsilon_d|^2\,] = 2\,\sigma_d^2 = 2\,\frac{2^{-2B}}{12} = \frac{2^{-2B}}{6}$$

Next, we derive approximate formulae for the upper and lower bounds on the root-mean-square error.

4.3.2.1 UPPER BOUND ANALYSIS

Linear Interpolation:

The upper bound of the error is obtained by assuming that during each addition in the computation scheme there is an overflow and a scaling is performed. In the following analysis, we will neglect the effects of A/D conversion noise. It can be seen from Fig 4.3(b) that the complex errors add directly to the output.

$$\text{For } n \le \frac{M}{2} \quad \mathrm{Var}(S_n) = \frac{1}{2}[\log_2 n]\,(2\,\sigma_a^2) + (2\,\sigma_d^2)$$

$$n > \frac{M}{2} \quad \mathrm{Var}(S_n) = [\log_2 n]\,(2\,\sigma_a^2) + (n-1)\,(2\,\sigma_m^2) + (2\,\sigma_d^2) \qquad (4.7)$$

Note that the second expression is a conservative estimate since for some n, which divide N, there are no multiplications involved in computing S_n. Furthermore, some of the multiplications and divisions can be done without error. The variance of a_1 is therefore

$$\mathrm{Var}(a_1) < \sum_{n=1}^{M} \mathrm{Var}(S_n)$$

$$= \sum_{n=1}^{M/2} \{ \frac{1}{2} [\log_2 n] (2\sigma_a^2) \} + \sum_{n=M/2+1}^{M} \{ [\log_2 n] (2\sigma_a^2) \}$$

$$+ \sum_{n=M/2+1}^{M} (n-1)(2\sigma_m^2) + \sum_{n=1}^{M} (2\sigma_d^2)$$

$$\approx \frac{(2\sigma_a^2)}{\ln(2)} \{ M \ln M - M + 1 \} + \frac{M}{4} (\frac{3M}{2} - 1)(2\sigma_m^2) + M(2\sigma_d^2)$$

$$= \frac{2^{-2B}}{\ln(2)} \{ M \ln M - M + 1 \} + \frac{M}{24} (\frac{3M}{2} - 1) 2^{-2B} + \frac{M}{6} 2^{-2B}$$

Now if we let the rms of the completed transform be equal to $|a_1|$, then the ratio of the rms of the error to the rms of the completed transform is

$$\frac{\mathrm{rms(error)}}{\mathrm{rms(result)}} = \frac{2^{-B} \sqrt{\frac{1}{\ln(2)} \{ M \ln M - M + 1 \} + \frac{M}{24} (\frac{3M}{2} - 1) + \frac{M}{6}}}{|a_1|} \qquad (4.8)$$

The upper bound for the error is proportional to M, the output transform size. That is, if M is doubled, then to maintain the same ratio of the rms of the error to the rms of the result, one bit must be added to the register length.

Zero Order Interpolation:

The upper bound for the error in the AFT implementation using zero order interpolation can be obtained by discarding the error term due to

multiplication in Equation (4.8). Therefore, the ratio for the rms of the error to the rms of the completed transform is given by:

$$\frac{\text{rms(error)}}{\text{rms(result)}} = \frac{2^{-B}\sqrt{\frac{1}{\ln(2)}\{M \ln M - M + 1\} + \frac{M}{6}}}{|a_1|} \qquad (4.9)$$

The upper bound for the error is proportional to $\sqrt{M}$, i.e., if M is quadrupled, then to maintain the same ratio, one bit must be added to the register length.

4.3.2.2 LOWER BOUND ANALYSIS

Linear Interpolation:

The lower bound for the error is obtained by assuming that there are no overflows resulting from addition operations and, hence, no scaling or shift is required. Then, the first term of (4.8) vanishes, and we are left with the lower bound for the ratio of the rms of the error to the rms of the completed transform:

$$\frac{\text{rms(error)}}{\text{rms(result)}} = \frac{2^{-B}\sqrt{\frac{M}{24}(\frac{3M}{2}-1) + \frac{M}{6}}}{|a_1|} \qquad (4.10)$$

The lower bound for the error is proportional to M. We notice that in Equations (4.8) and (4.10), the dominating error term is due to multiplication. The fact that we assumed that the calculation of the Riemann sums for $n > \frac{M}{2}$ all involve linear interpolation of the worst scenario will not affect the output error functional dependence on M. However, a more realistic estimate will involve a weaker proportionality constant.

To ensure that there are no overflows is to have the average modulus squared of the initial signal small and, in fact, one which approaches zero as N becomes large. In a practical situation, one requires $|g_i|<1$ for i=0,1,2,...,N-1, and incorporating an attenuation of 1/2 at the input of each stage of Fig 4.3(b). In this case the output will consist not of S_n but of $\frac{1}{2n}$ times this S_n.

Zero Order Interpolation:

Removing the error term due to multiplication from Equation (4.10), we obtain the lower bound for the error using zero order interpolation

$$\frac{\text{rms(error)}}{\text{rms(result)}} = \frac{2^{-B}\sqrt{\frac{M}{6}}}{|a_1|} \qquad (4.11)$$

The lower bound for the error is proportional to $\sqrt{M}$. Notice that Equation (4.11) contains the error caused by the division operations, which is expected if proper scaling of the input data and zero order interpolation procedure is used.

In Fig 4.4, we plot as a function of the output transform size, M, the approximate upper and lower bounds for the ratio of the rms of the error to the rms of the completed transform ($|a_1|=1$) using zero order and linear interpolation procedures for B=14 and B=17. Based on this analysis, for architectures I, we suggest that the word size for the representation of the initial input signal, B, should not be less than 14 if a reasonable error bound is desired. Notice that in our derivations of the upper and lower bound for the rms of the error, we assumed that the linear interpolants are represented in the same way as the input signal with B bits. Clearly, this assumption is an absolutely pessimistic bound for multiplier representation for small and

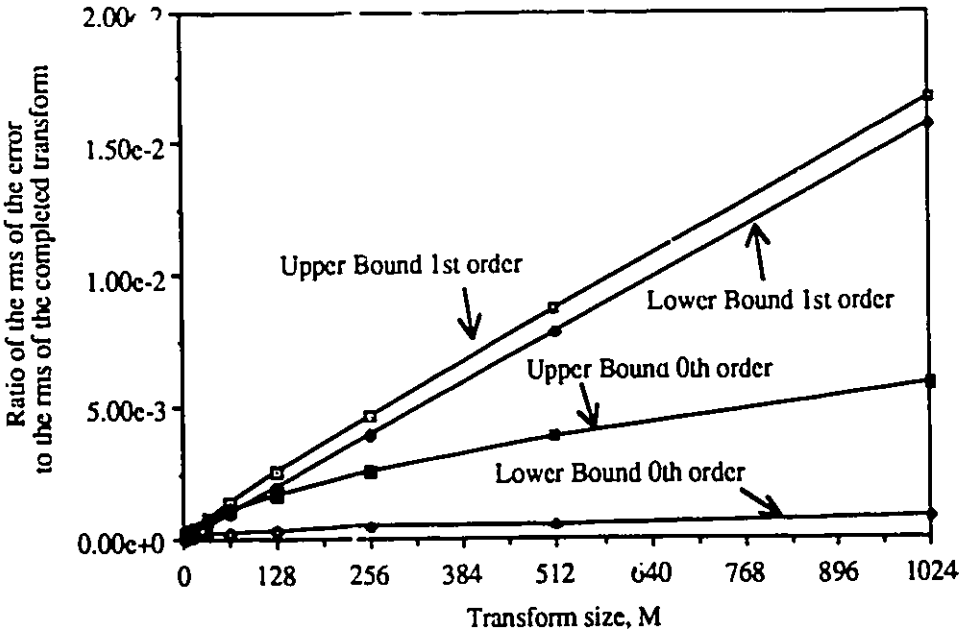


Fig 4.4(a) : Quantization noise for architecture I; B=14 bits.

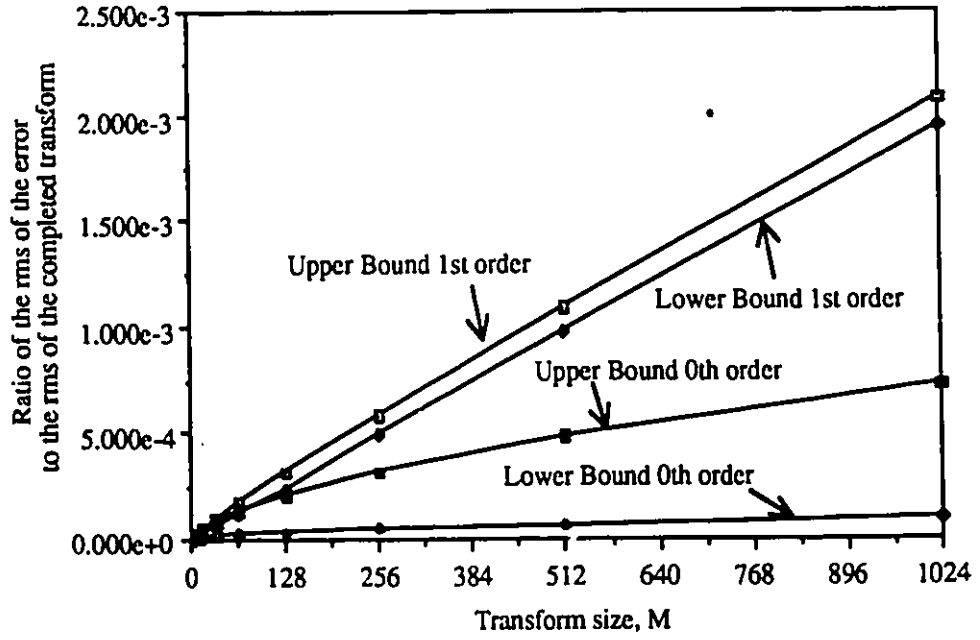


Fig 4.4(b) : Quantization noise for architecture I; B=17 bits.

moderate transform sizes. Moreover, in the derivation we assumed that in the latter half of { S_i }, the order, i, is a prime number so that the error behaves in the worst case scenario. Of course, this assumption is very conservative, and one that leads to a pessimistic bound for the error estimate. Notice that linear interpolation results in larger quantization noise than zero order interpolation. This is due to the fact that linear interpolation involves multiplication and consequently rounding noise and zero order interpolation does not. However, as we have seen in chapter 2, zero order interpolation results in larger sample interpolation error which is generally larger than the quantization error. It seems that there is a tradeoff between using linear interpolation (smaller sample interpolation error and larger quantization error) and zero order interpolation (larger sample interpolation error and smaller quantization error) for implementing the AFT.

4.3.3 CRITICAL PATH ANALYSIS

One of the important considerations in implementing the AFT is the estimation of the longest computational path or critical path for a non-pipelined system. Let τ_s, τ_a, τ_m, and τ_d be the unit of time required to execute, respectively, a shift operation, an addition operation, a multiplication operation, and a division operation. Assume that the computations start when all samples are correctly positioned in the right hand N/2-stage shift register. We will also assume that the multiplications and divisions are each performed in parallel, and that the additions are cascaded in a tree structure. Then the critical path (CP) is given by (the starting point is considered to be the output of the input subtractor):

$$CP = \frac{N}{2}(\tau_s+\tau_a) + \tau_m + \log_2(M-2)\,\tau_a + \tau_d + \log_2(M/2)\,\tau_a$$

$$= \frac{N}{2}\tau_s + [\frac{N}{2} + \log_2\frac{M}{2}(M-2)]\,\tau_a + \tau_m + \tau_d \qquad (4.12)$$

For the zero order case, the critical path is given by

$$CP = \frac{N}{2}\tau_s + [\frac{N}{2} + 2\log_2(M/2)]\,\tau_a + \tau_d \qquad (4.13)$$

In both interpolation schemes, the critical path is proportional to N, the length of the input sequence. As an example, consider the 12 coefficient, 60 sample system. It is reasonable to assume $\tau_m = \tau_d = 2\tau_a = 8\tau_s = \tau$ for a given word size, B, and 3μ double-metal CMOS technology. Then the CP for linear and zero order interpolation are, respectively, 24τ and 23τ. Notice that we have assumed the multiplications for each S_i to be done in parallel, and consequently they will take one unit of time, τ_m, to perform.

4.4 SIMULATION OF ARCHITECTURE II

We will consider implementing the linear interpolation scheme for symmetric functions. The zero order implementation will be a simple subset with both summed sample pairs and multiplier reduction.

The simulation block diagram for hardware implementation using architecture II is shown in Fig 4.5. The results of the simulation are given in Appendix E. We used the same input signal as in the previous simulation. Again the arithmetic elements are not pipelined in this simulation. The total hardware requirement for this implementation is as follows:

1 Multiplier/Accumulator with divider. This is used to obtain the mean value of the input sequence

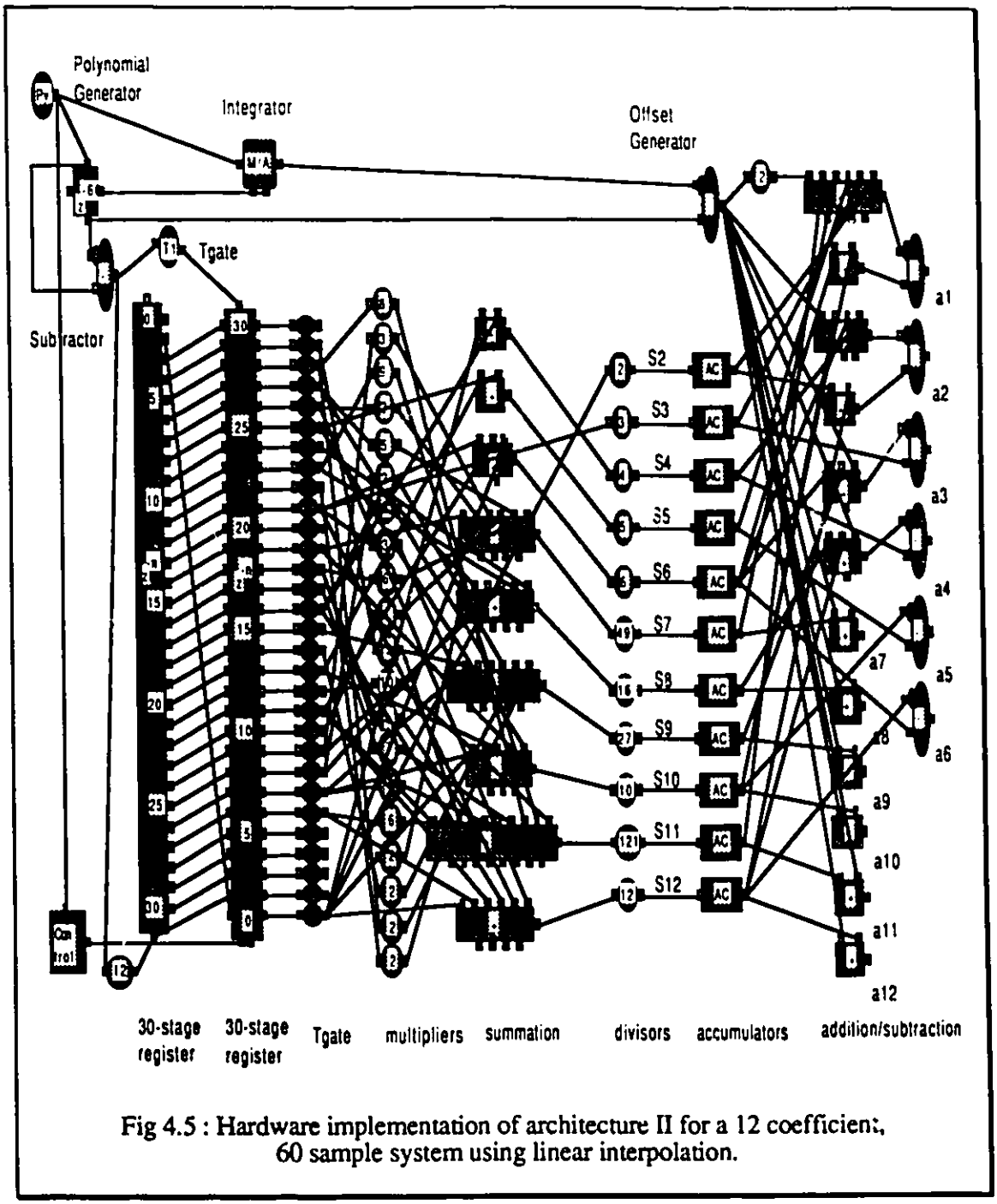


Fig 4.5 : Hardware implementation of architecture II for a 12 coefficient, 60 sample system using linear interpolation.

2 30 stage shift registers with parallel load capability

11 Accumulators

21 multipliers (maximum multiplicand is 10)

11 dividers

58 2-input adders

8 2-input subtractors

The transmission gates, T1 and T2 at the registers input, are used as switches to control the flow of the input sequence. In Fig 4.5, the N/2 transmission gates at the output of the right hand 30-stage shift register are not needed in a silicon realization. However, they are used here to allow the data to pass at each N/2 time interval. The accumulators are used to sum each consecutive half period results, as explained in chapter 3.

The major difference between this implementation and the implementation of architecture I is in terms of speed. Because of reduced data transfer and elimination of the adder in the register input, the throughput of architecture II will be higher than that of architecture I. Moreover, architecture II will have the ability to produce Fourier coefficients at every consecutive half period. In terms of hardware, architecture II replaces the adder/right hand 30-stage shift register combination with 11 accumulators.

4.4.1 DYNAMIC RANGE GROWTH

The dynamic range growth (DRG) for this architecture is the same as for the previous one (it is noted that the summation of the input pairs in architecture I is compensated for by the accumulator stage in architecture II).

4.4.2 QUANTIZATION ANALYSIS

The fixed-point accuracy of implementing the AFT using architecture II is similar to that of architecture I. Therefore, the formulae for the ratio for the rms of the error to the rms of the completed transform derived for the previous architecture apply to this architecture.

4.4.3 CRITICAL PATH ANALYSIS

Here we will also assume that the multiplications and divisions are performed in parallel, and the additions are cascaded in a tree structure. If we assume the starting point to be the output of the input subtractor, then the critical path for the linear interpolation case is given by:

$$CP \approx \frac{N}{2}\tau_s + \tau_m + \log_2(M\text{-}2)\,\tau_a + \tau_d + \tau_a + \log_2(M/2)\,\tau_a$$

$$= \frac{N}{2}\tau_s + [1+ \log_2\frac{M}{2}(M\text{-}2)]\,\tau_a + \tau_m + \tau_d \qquad (4.14)$$

For the zero order case, the critical path is given by

$$CP \approx \frac{N}{2}\tau_s + [1 + 2\log_2(M/2)]\,\tau_a + \tau_d \qquad (4.15)$$

The critical path is proportional to $\log_2 M$(for the addition operations) for both zero order and linear interpolation schemes. Notice that the CP for this architecture is smaller than that for the previous one by an amount of $\frac{(N\text{-}1)}{2}\tau_a$. For the 12 coefficient, 60 sample example, the CP for linear and zero order interpolation schemes are, respectively, 9τ and 8τ, a reduction of 15τ over the previous architecture.

4.5 SIMULATION OF ARCHITECTURE III

We will consider implementing the linear interpolation scheme for general functions using Type(a) geometry (Fig 3.3(a)). The simulation block diagram for hardware implementation using architecture III (Type(a) geometry) is shown in Fig 4.6. The results of the simulation are given in Appendix E. We used the same input signal as in the previous simulations. The total hardware requirement is as follows:

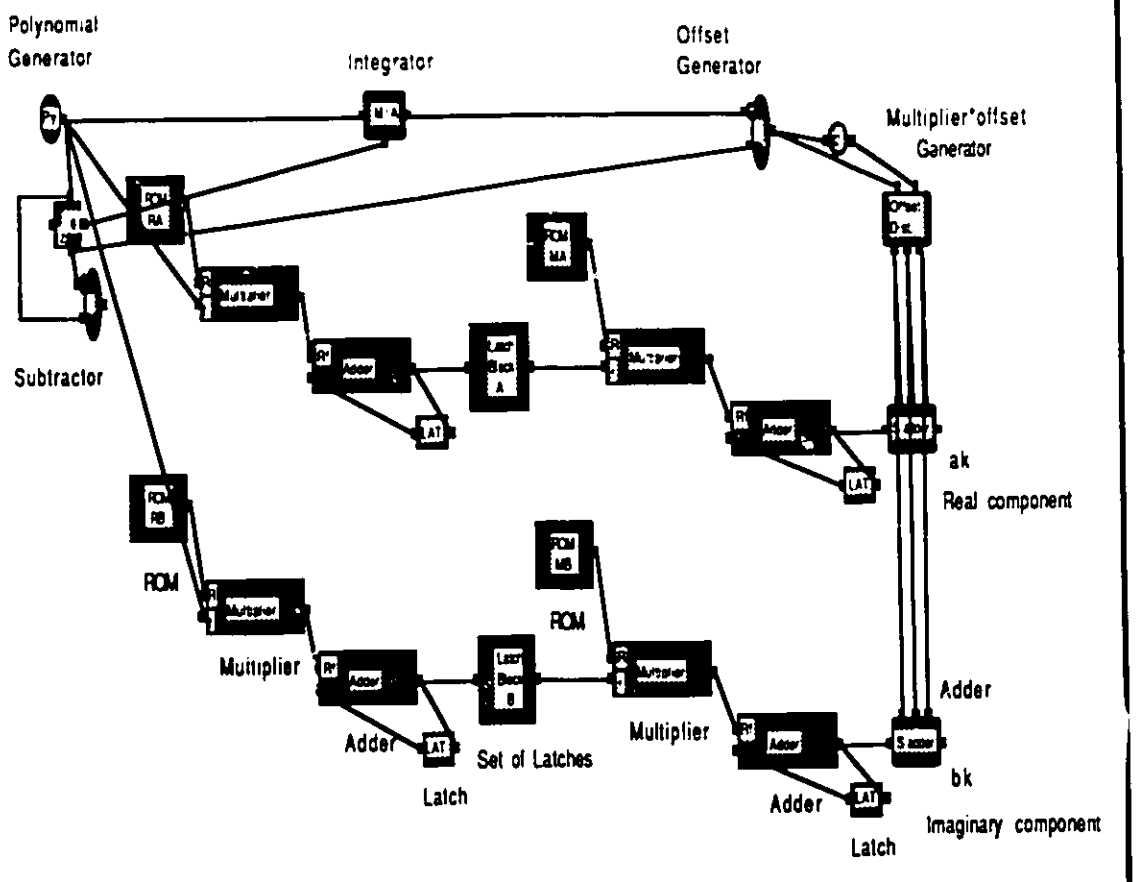


Fig 4.6 : Hardware implementation of architecture III(Type(a) geometry) for a 12 coefficient, 60 sample system using linear interpolation.

1 Multiplier/Accumulator with divider. This is used to obtain the mean value of the input sequence

1 ROM (MxN words; Max. entry < 12)
(For pre-storing the entries of the R matrix for calculating a_k's)

1 ROM (MxM words)
(For pre-storing the entries of the U matrix for calculating a_k's)

1 ROM (M1xN words; Max. entry < 12)
(For pre-storing the entries of the R matrix for calculating b_k's)

1 ROM (MxM1 words)
(For pre-storing the entries of the U matrix for calculating b_k's)

4 multipliers

6 2-input adders

2 2-input subtractors

The divisors are built-in to the U matrix, as explained in chapter 3. That is, the j-th column of U is divided by the divisor of the j-th Riemann sum. The multiplier-offset generator is used to generate the three different weighted offsets: $(g_0 - \bar{g})$; $-(g_0 - \bar{g})$; and $-2(g_0 - \bar{g})$ needed to implement Table 3.4. Since the Fourier coefficients are computed sequentially through one output node the incorporation of these offsets to the scheme of computation is more elaborate in this architecture than in the previous ones, and therefore care must be taken to synchronize this operation.

4.5.1 DYNAMIC RANGE GROWTH

A typical calculation using this architecture is shown in Fig 4.7. Again the maximum multiplier is M-2 (assuming M-1 to be a prime number). The maximum number of multipliers involved in computing one Fourier coefficient is 2M (this is the maximum number of linear interpolants in one row of the R matrix). If we assume an average multiplier of (M-2)/2, then the maximum DRG due to multiplication is $2M \log_2(\frac{M}{2} - 1)$. The maximum DRG due to addition is (2M - 1). The output summation stage has maximum DRG of M. The combination is a worst case of

$$DRG \approx 2M \log_2(\frac{M}{2} - 1) + 3M - 1 \qquad (4.16)$$

In this case, the DRG is proportional to M, the output transform size. For the 12 coefficient example, the DRG is 91 bits. This is the maximum dynamic range growth for computation of both the real and imaginary components of the AFT. For the zero order case, the DRG is given by

$$DRG = 3M - 1 \qquad (4.17)$$

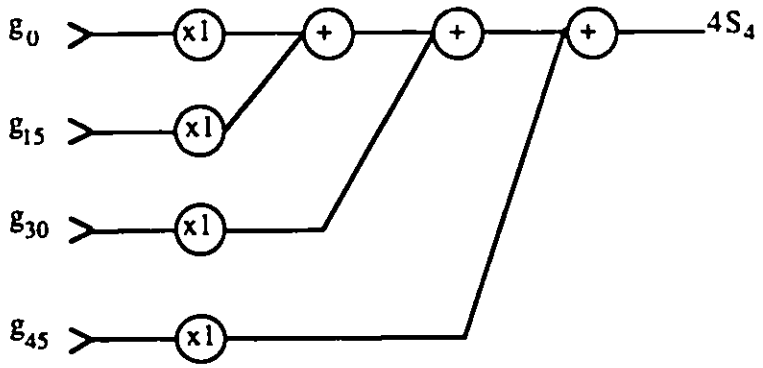


Fig 4.7 : Typical calculation in architecture III.

For the example above, the zero order interpolation gives maximum DRG of 35 bits.

4.5.2 QUANTIZATION ANALYSIS

The style of analysis illustrated by the previous discussion can be applied to investigate the effects of quantization in architecture III.

4.5.2.1 UPPER BOUND ANALYSIS

Linear Interpolation:

The computation of the Riemann sums using architecture III is defined by

$$\mathbf{S} = \mathbf{R}\,\mathbf{g} \qquad (4.18)$$

where **g** is an Nx1 vector containing the input sequence, R is an MxN matrix whose entries are the linear interpolants, and **S** is an Mx1 vector containing the Riemann sums sequence. The flow graph for a typical calculation using this architecture is shown in Fig 4.8(a). Fig 4.8(b) shows the same structure with noise sources added. The assumptions and simplifications made in the previous discussion still hold. Let A_1 designate either a_1 or b_1. Then the variance of A_1 is

$$\begin{aligned} \mathrm{Var}(A_1) &< \sum_{n=1}^{M} \mathrm{Var}(S_n) \\ &< \sum_{n=1}^{M} 2n\,(2\sigma_a^2) + \sum_{n=M/2+1}^{M} 2n\,(2\sigma_m^2) + \sum_{n=1}^{M} (2\sigma_d^2) \\ &= M(M+1)\,2^{-2B} + \frac{M}{12}\left(\frac{3M}{2}+1\right)2^{-2B} + \frac{M}{6}\,2^{-2B} \end{aligned}$$

The ratio of the rms of the error to the rms of the completed transform is therefore

$$\frac{\mathrm{rms(error)}}{\mathrm{rms(result)}} = \frac{2^{-B}\sqrt{M(M+1) + \frac{M}{12}\left(\frac{3M}{2}+1\right) + \frac{M}{6}}}{|A_1|} \qquad (4.19)$$

The upper bound of the error is proportional to M.

Zero Order Interpolation:

The upper bound for the error using zero order interpolation is

$$\frac{\mathrm{rms(error)}}{\mathrm{rms(result)}} = \frac{2^{-B}\sqrt{M(M+1) + \frac{M}{6}}}{|A_1|} \qquad (4.20)$$

Thus the output error for the zero order case is also proportional to M.

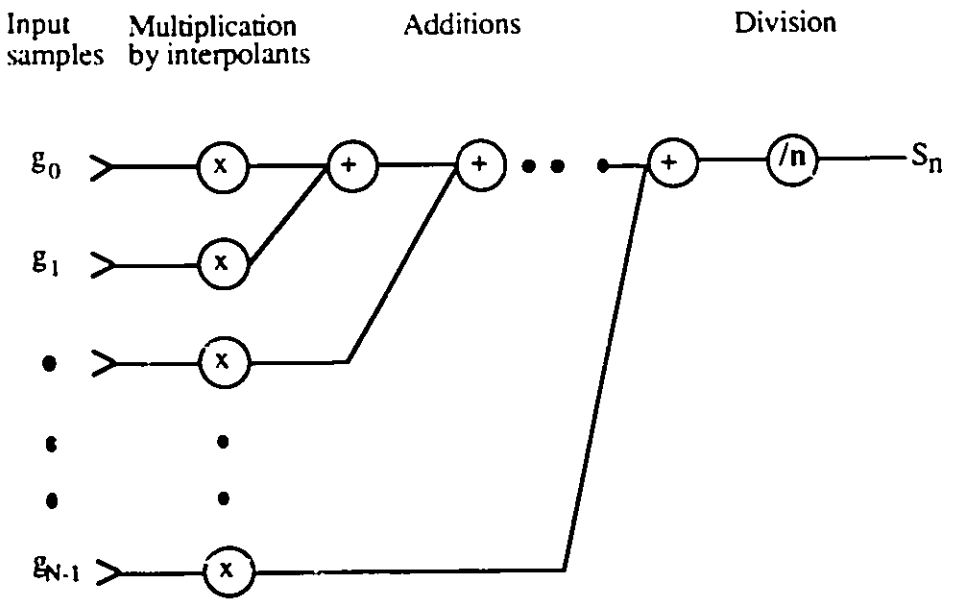


Fig 4.8(a) : Flow graph for architecture III.

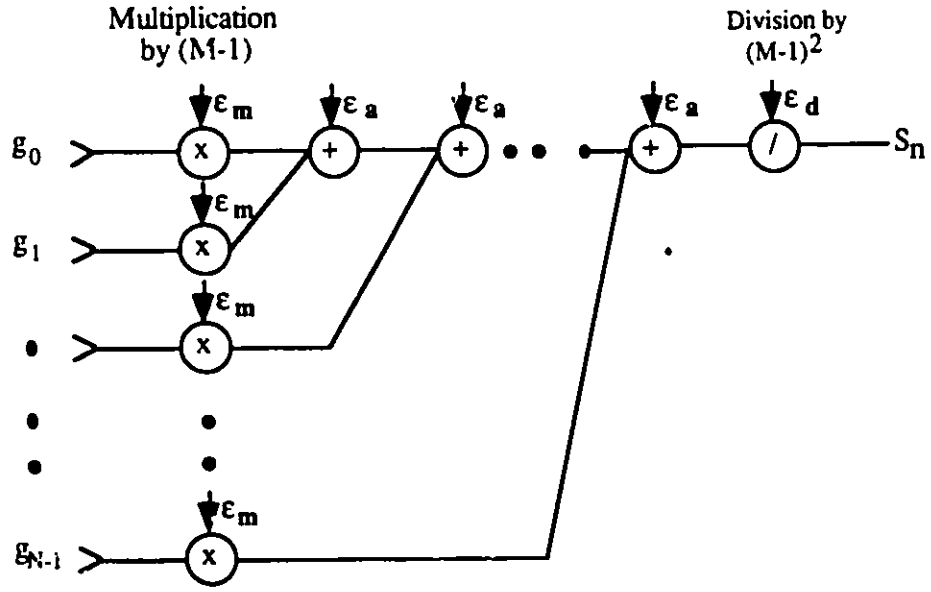


Fig 4.8(b) : Flow graph with noise for architectures III.

4.5.2.2 LOWER BOUND ANALYSIS

Linear Interpolation:

The lower bound for the error is obtained by assuming that there are

no overflows resulting from addition operations. Hence the first term of Equation (4.19) vanishes, and the ratio of the rms of the error to the rms of the completed transform is

$$\frac{\text{rms(error)}}{\text{rms(result)}} = \frac{2^{-B}\sqrt{\frac{M}{12}(\frac{3M}{2}+1)+\frac{M}{6}}}{|A_1|} \qquad (4.21)$$

The lower bound of the error is proportional to M, the output transform size.

Zero Order Interpolation:

The lower bound for the error using zero order interpolation is

$$\frac{\text{rms(error)}}{\text{rms(result)}} = \frac{2^{-B}\sqrt{\frac{M}{6}}}{|A_1|} \qquad (4.22)$$

The output error for the zero order interpolation procedure is proportional to $\sqrt{M}$.

In Fig 4.9, we plot as a function of the output transform size, M, the approximate upper and lower bounds for the ratio of the rms of the error to the rms of the completed transform ($|A_1|=1$) using zero order and linear interpolation procedures for B=14 and B=17. Based on this analysis, for architecture III, we suggest that the word size for the representation of the initial input signal, B, should not be less that 14 if a reasonable error bound is desired. Again, we stress the fact that the analysis is based on a very conservative and an absolutely pessimistic bound for multiplier representation for small and moderate AFT sizes.

Comparison between this architecture and the previous architectures indicates that the ratio of the rms of the error to the rms of the completed transform for architecture III is approximately 4 times larger than that of architectures I and II.

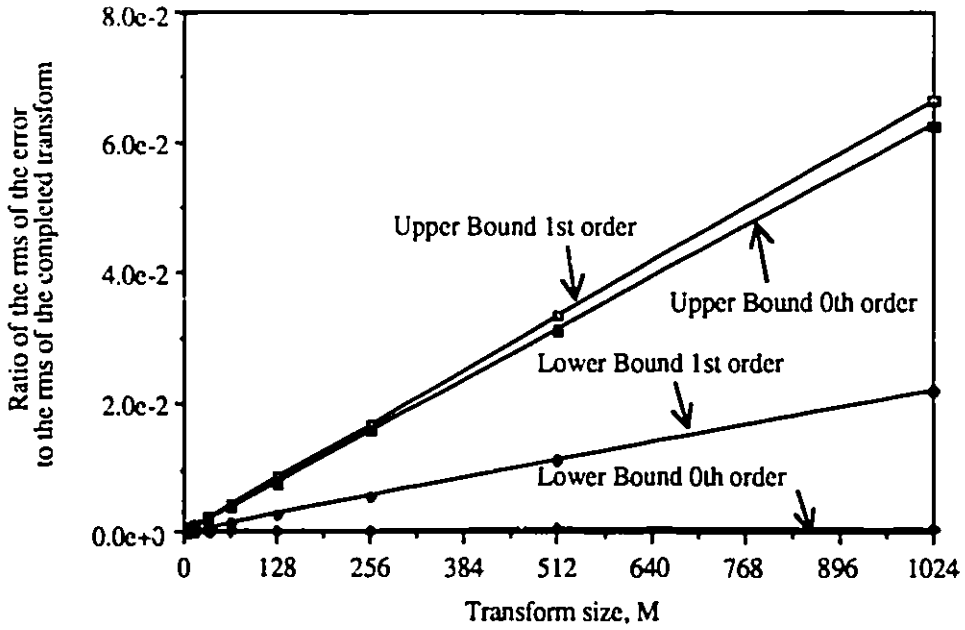


Fig 4.9(a) : Quantization noise for architecture III; B=14 bits.

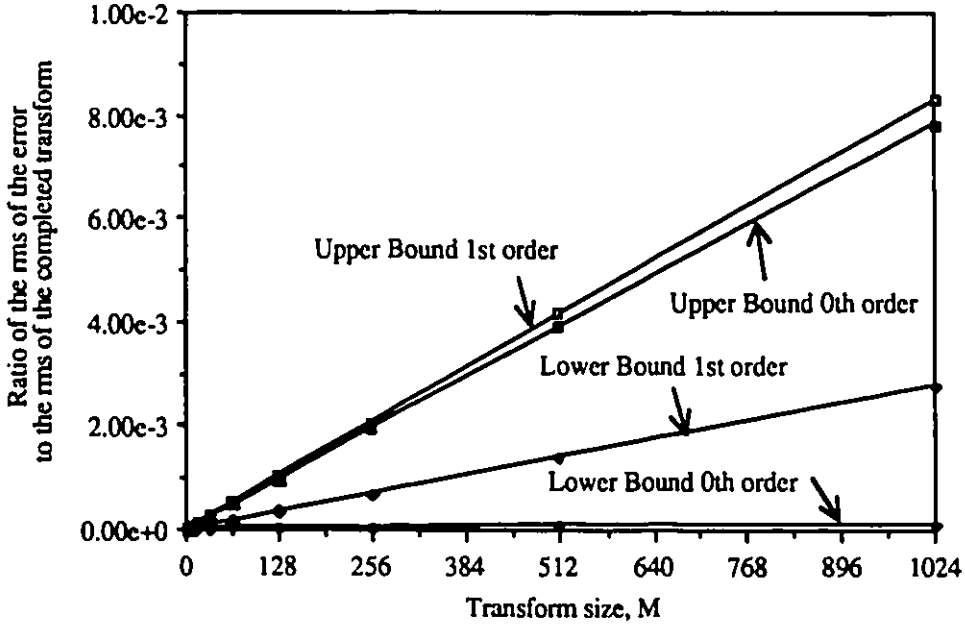


Fig 4.9(a) : Quantization noise for architecture III; B=17 bits.

4.5.3 CRITICAL PATH ANALYSIS

We will assume that the computation of the Fourier coefficients are done in parallel for optimum speed.. This means that to compute M Fourier coefficients, the configuration of Fig 4.6 must be replicated M times in parallel. If we assume that the waiting cycle for memory access time is zero, then the critical path is given by

$$CP \approx 2(M-2)(\tau_m + \tau_a) + M(\tau_m + \tau_a)$$

$$= (3M-4)(\tau_m + \tau_a) \qquad (4.23)$$

The first term is due to computation of the Riemann sums and the second term is due to computation of the Fourier coefficients. For the zero order case, the CP is

$$CP \approx M\,\tau_a + M(\tau_m + \tau_a) = M(\tau_m + 2\,\tau_a) \qquad (4.24)$$

In both interpolation procedures, the critical path is proportional to the output transform size, M. For the 12 coefficient example, the CP for linear and zero order interpolation schemes are, respectively, 48τ and 24τ.

4.6 SIMULATION OF ARCHITECTURE IV

We will consider implementing the zero order interpolation scheme for computing 12 complex Fourier coefficients from 60 samples. The simulation block diagram for hardware implementation using architecture IV is shown in Fig 4.10. The results of the simulation are given in Appendix E. We used the same input signal as in the previous simulations. The total hardware requirement for this implementation is as follows:

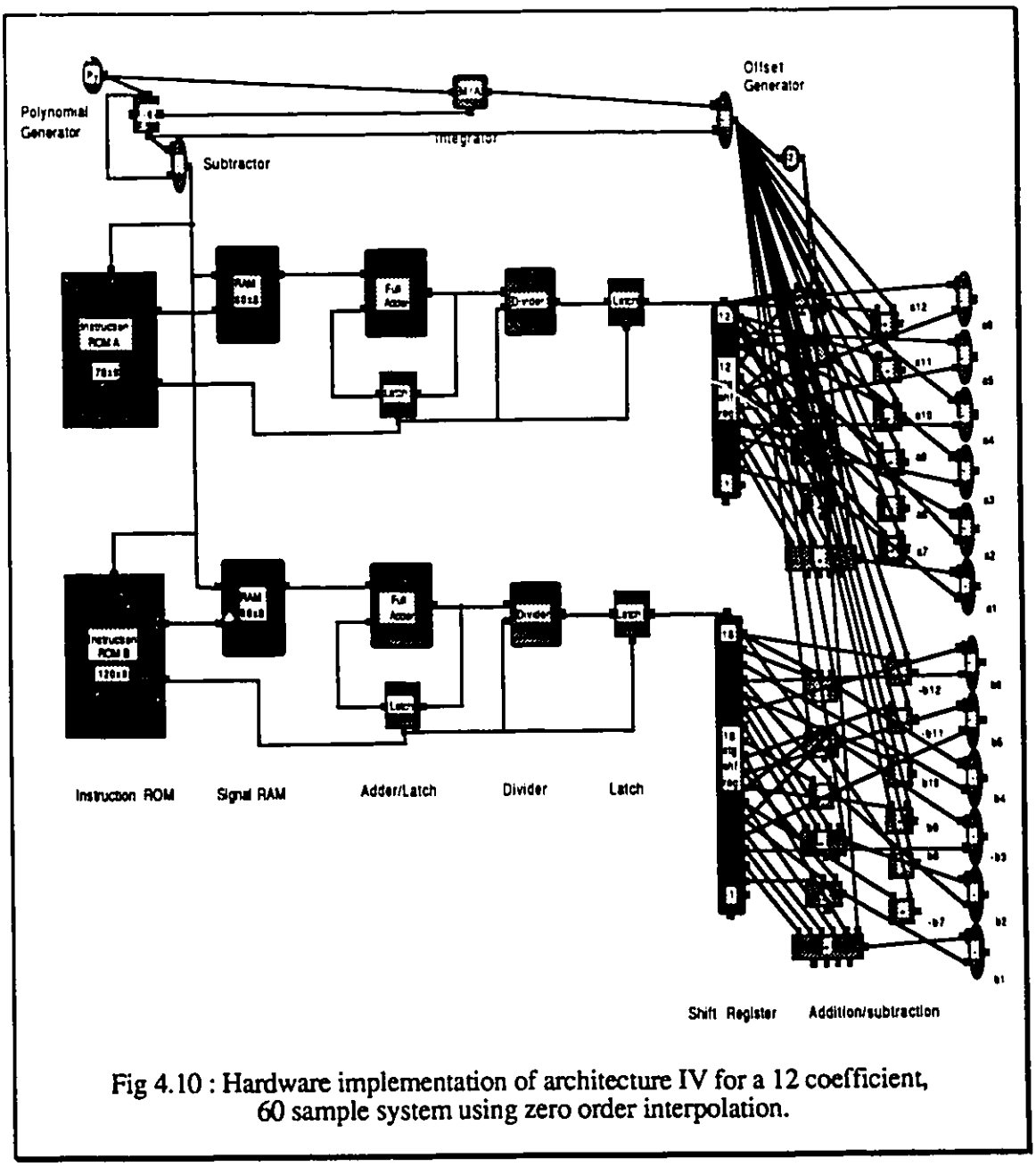


Fig 4.10 : Hardware implementation of architecture IV for a 12 coefficient, 60 sample system using zero order interpolation.

1 Multiplier/Accumulator with divider. This is used to obtain the mean value of the input sequence

1 ROM (78x6)

1 ROM (120x7)

2 RAMs (60x8)

2 dividers

1 12 stage shift register

1 18 stage shift register

38 2-input adders

14 2-input subtractors

In a silicon realization, the last four components of the hardware will be replaced with a RAM, to store the values of the Riemann sums, and an accumulator with add/subtract capability to generate the Fourier coefficients from linear combinations of Riemann sums, as discussed in chapter 3. The memory sizes required for this example are calculated in chapter 3. The division operation can be implemented by using either a digital divider or a look-up table.

4.6.1 DYNAMIC RANGE GROWTH

A typical calculation in the implementation of Fig 4.10 is shown in Fig 4.11. Since the implementation is based on the zero order interpolation procedure, there are no multipliers involved in the computation. Hence, the dynamic range growth will be solely due to addition operations. According to Fig 4.11, the maximum DRG is M, the order of the largest Riemann sum. If we assume that the output summation stage is implemented with a RAM and an accumulator, then the output summation stage has maximum DRG of M. The combination is a worst case of

$$DRG \approx 2M \qquad (4.25)$$

The maximum DRG is proportional to the output transform size, M. Equation (4.25) is valid for the computation of both the real and imaginary

components of the AFT. For the 12 coefficient, 60 sample example, the maximum DRG will be 24 bits.

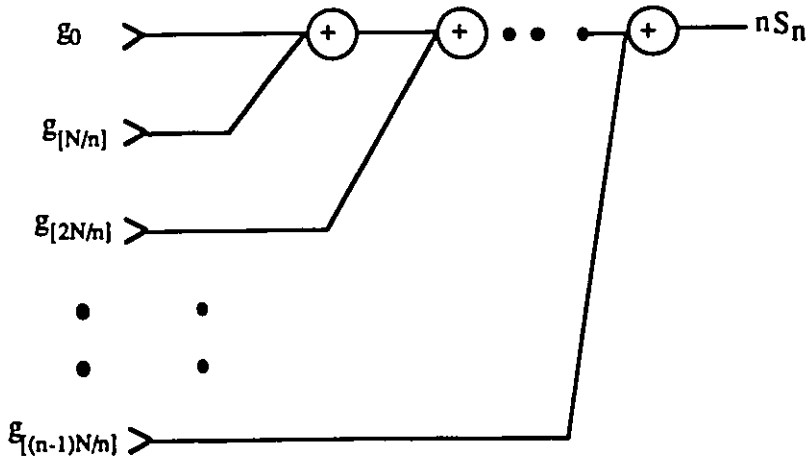


Fig 4.11: Typical calculation of a Riemann sum using architecture IV.

4.6.2 QUANTIZATION ANALYSIS

We will consider here the AFT implementation using zero order interpolation.

4.6.2.1 UPPER BOUND ANALYSIS

The flow graph for a typical calculation using architectures IV is shown in Fig 4.12(a). Fig 4.12(b) shows the same structure with noise sources added. The assumptions and simplifications made in the previous discussions still hold. Let A_1 designate either a_1 or b_1. Then the variance of A_1 is

$$\mathrm{Var}(A_1) < \sum_{n=1}^{M} \mathrm{Var}(S_n)$$

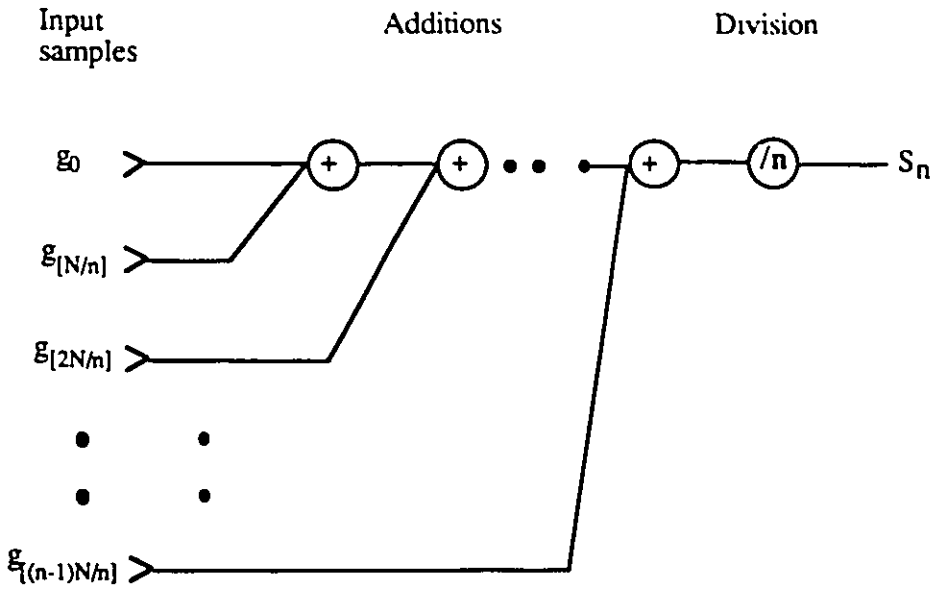


Fig 4.12(a) : Flow graph for architecture IV.

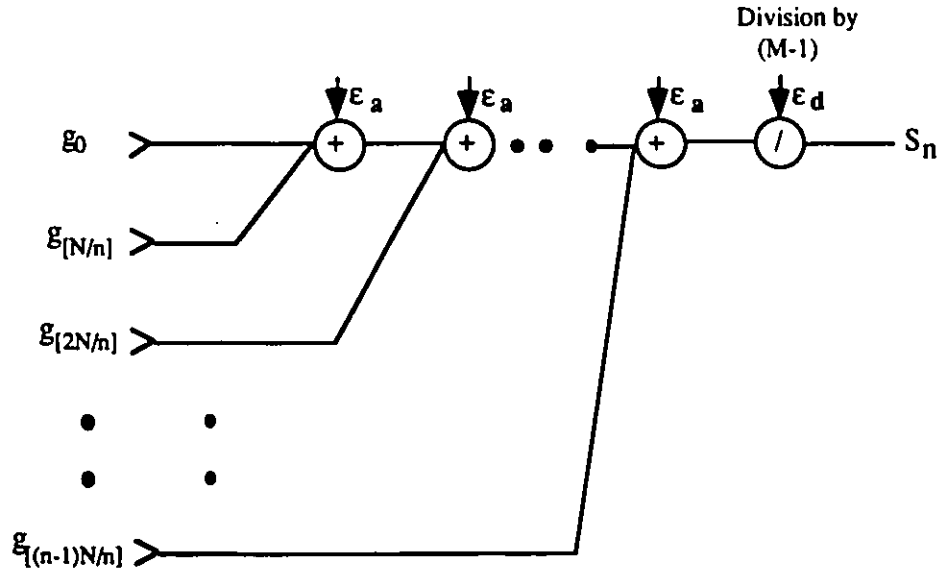


Fig 4.12(b) : Flow graph with noise added for architecture IV.

$$= \sum_{n=1}^{M} (n-1)\,(2\,\sigma_a^2) + \sum_{n=1}^{M} (2\,\sigma_d^2)$$

$$= \frac{M}{2}(M-1)\,2^{-2B} + \frac{M}{6}\,2^{-2B}$$

The ratio of the rms of the error to the rms of the completed transform is therefore

$$\frac{\text{rms(error)}}{\text{rms(result)}} = \frac{2^{-B}\sqrt{\frac{M}{2}(M-1)+\frac{M}{6}}}{|A_1|} \qquad (4.26)$$

The upper bound of the error is proportional to M, the output transform size.

4.6.2.2 LOWER BOUND ANALYSIS

The lower bound of the error is obtained by assuming there are no overflows, and consequently the first term of Equation (4.15) vanishes

$$\frac{\text{rms(error)}}{\text{rms(result)}} = \frac{2^{-B}\sqrt{\frac{M}{6}}}{|A_1|} \qquad (4.27)$$

Again, the lower bound of the error is proportional to $\sqrt{M}$, when there are no overflows and zero order interpolation is used.

In Fig 4.13, we plot as a function of the output transform size, M, the approximate upper and lower bounds for the ratio of the rms of the error to the rms of the completed transform ($|A_1|=1$) using zero order procedure for B=14 and B=17. Based on this analysis, for architectures IV, we suggest that the word size for the representation of the initial input signal, B, should not be less that 14 if a reasonable error bound is desired.

Comparison between this architecture and the previous architectures for the zero order interpolation procedure indicates that architecture IV has approximately the same magnitude of error as architecture III, but 4 times larger than architectures I and II. Note that the lower bound of the error is exactly the same in all architectures.

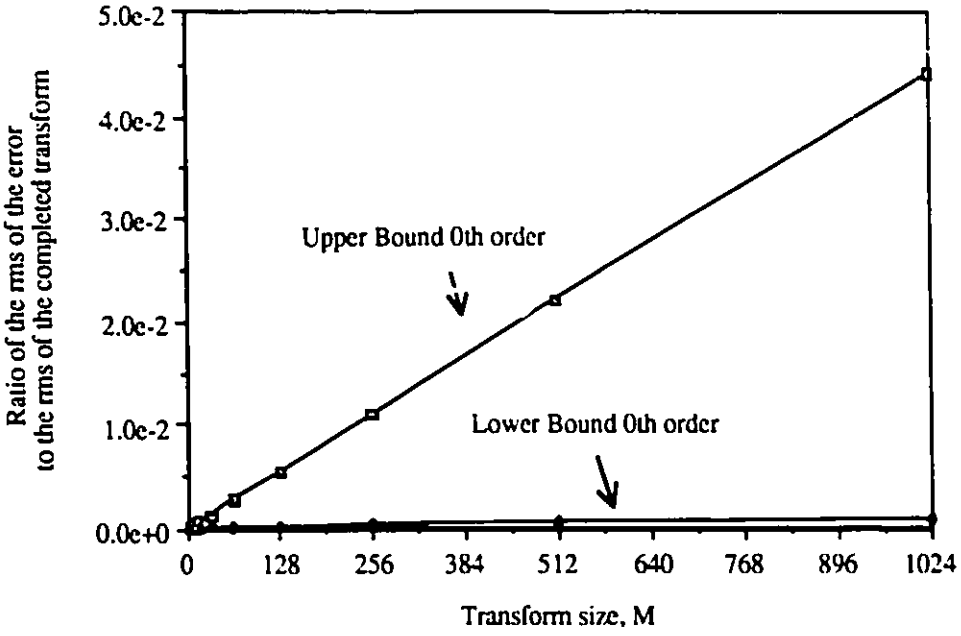


Fig 4.13(a): Quantization noise for architecture IV; B=14 bits.

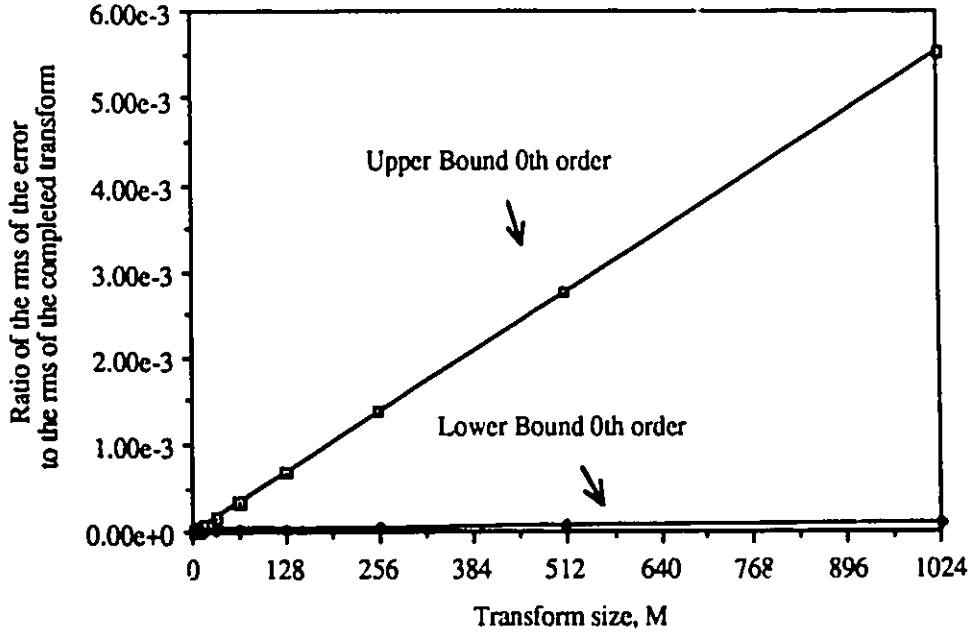


Fig 4.13(b) : Quantization noise for architecture IV; B=17 bits.

4.6.3 CRITICAL PATH ANALYSIS

The implementation of architecture IV computes the real and imaginary components of the AFT in sequ[illegible]e longest computational path is involved in the calculation of the first component of the AFT. If we assume the memory access time to be zero, then the critical path is given by

$$CP = \sum_{k=1}^{M} (k-1)\,\tau_a + M1\tau_d + \sum_{k=1}^{M/2} \sum_{i=1}^{[M/k]} |\mu(i)|\,\tau_a$$

$$< [\frac{M}{2}(M-1) + M \ln(M/2)]\,\tau_a + M1\tau_d \qquad (4.28)$$

The critical path is proportional to M^2 for the addition operations and proportional to M1 for the division operations, where M1 is the number of Riemann sums needed for the computation of the imaginary component of the AFT. For the 12 coefficient example, the CP is 62τ.

4.7 SIMULATION OF ARCHITECTURE V

We will consider implementing the zero order interpolation scheme for symmetric functions (although this architecture can be used to compute the AFT for general functions). The simulation block diagram for computing 12 Fourier coefficients from 60 samples using architecture V is shown in Fig 4.14. The results of the simulation are given in Appendix E. We used the same input signal as in the previous simulations. The total hardware requirement for this implementation is :

1 Multiplier/Accumulator with divider. This is used to obtain the mean value of the input sequence

16 Controlled switches

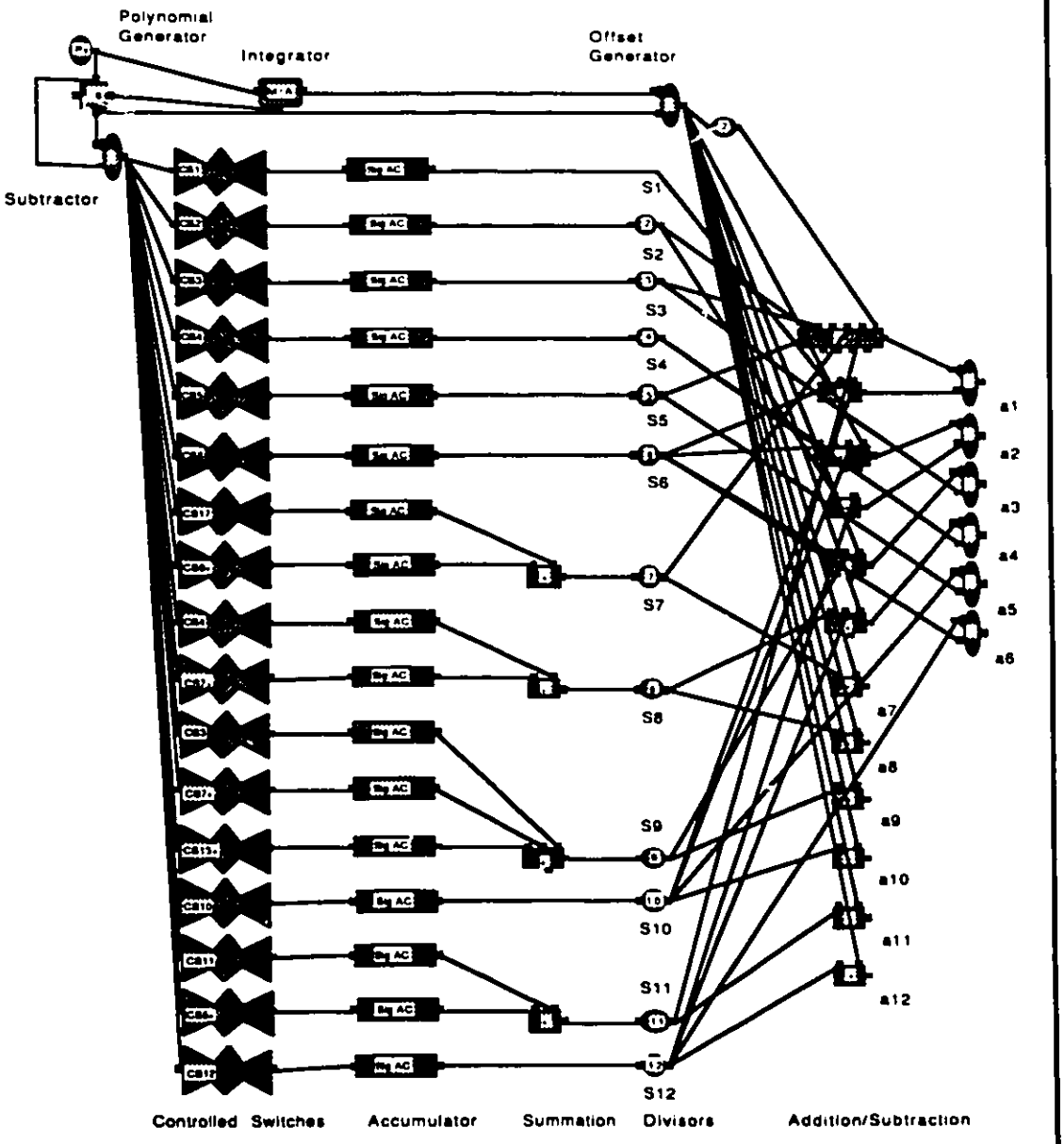


Fig 4.14 : Hardware implementation of architecture V for a 12 coefficient, 60 sample system using zero order interpolation.

16 Accumulators

11 dividers

26 2-input adders

8 2-input subtractors

In a silicon realization the controlled switches can be implemented either by using a demultiplexer or by using a complex clocking scheme. For explanation of the operation of the architecture, refer to chapter 3.

4.7.1 DYNAMIC RANGE ANALYSIS

The calculation of a typical Riemann sum using this architecture is similar to Fig 4.11. Thus the maximum DRG is M. The output summation stage for this architecture has maximum DRG of $\log_2 M$. The combination is a worst case of

$$DRG \approx M + \log_2 M \qquad (4.29)$$

The maximum DRG is proportional to the output transform size, M. This formula is valid for the computation of both the real and imaginary components of the AFT. For the 12 coefficient example, the maximum DRG is 16 bits. This value is comparable to the DRG of architecture IV.

4.7.2 QUANTIZATION ANALYSIS

The fixed-point accuracy analysis of this architecture is similar to that of architecture IV.

4.7.3 CRITICAL PATH ANALYSIS

The longest computational path is involved in computing the first component of the AFT. Thus the critical path is

$$CP \approx M\,\tau_a + \tau_d + \log_2(M/2)\,\tau_a$$

$$= [M + \log_2(M/2)]\,\tau_a + \tau_d \qquad (4.30)$$

The critical path is proportional to M for the addition operations and constant for the division operations. For the example above, the CP is 9τ.

4.8 COMPARISON BETWEEN THE ARCHITECTURES

In this section, we compare the proposed architectures in terms of dynamic range growth (DRG), quantization noise (QN) due to finite wordlength computation, critical path (CP), and hardware requirements. In terms of hardware requirements, the following observations are made:

(1) The hardware requirements of architectures I and II are dependent on the size of the input and output data as well as on the type of interpolation procedure used. architecture I is more so because it is based on the requirement of sample reversal to accomplish the alignment of table 3.2 and consequently two N/2-stage shift registers are needed. On the other hand, architecture II can perform with only one N/2-stage shift register to hold the input data.

(2) Architecture III can be easily modified to compute the Fourier coefficients of any size data as well as using either interpolation procedures. Of course, the internal control signal have to be modified to account for the input and output data sizes.

(3) Architecture IV is dependent on the size of the input and output data.

(4) Architecture V is independent of the size of the input data. It is fixed for a specific output transform size. This is an advantage because one can improve the accuracy of computation of the AFT without modifying the hardware itself. Of course, the internal clocks have to be modified to account for the appropriate samples to be used for the computation.

Table 4.1 shows a comparison in terms of DRG, QN, and CP. The values between parenthesis are for the 12 coefficient. 60 sample example. In the table, the relationships for quantization noise of architectures I, II, and III are for linear interpolation. The values for QN are computed using B=14 bits. The relations shown for the CP are concerning addition operation. For multiplication, the CP is constant except for architectures III and IV where the CP is proportional to, respectively, M and M1. The quantities between parenthesis for the critical path are calculated using the assumption of section 4.3.4.

It is clear from table 4.1 that architecture II is to be prefered over the other architectures since it has the smallest value for DRG; the smallest value for QN; and the shortest CP. Furthermore, architecture II has the ability to compute Fourier coefficients at each consecutive half period, as explained in chapter 3. However, in terms of hardware, architecture III provides an implementation with the least number of hardware components. Moreover, architecture III is the most versatile among the proposed architectures since the same hardware structure can be used for both zero order and linear interpolation procedures as well as for any AFT size. For example, to accommodate for different interpolation procedures, the memory structures used for pre-storing the constant coefficients have to be programmable ROMs or PROMs, and to accommodate for different AFT sizes, the PROMs have to be expandable. Another obvious advantage of architecture III is in terms of wire interconnections. There is no global, but local, interconnections involved in the implementation of architecture III, except for that part which involves the calculation of the mean of the signal. Note that architecture III can be implemented to compute the Fourier coefficients in a sequential order or in parallel as discussed in section 4.5.

	Dynamic Range Growth	Quantization Noise	Critical Path
Architecture I Symmetric Functions	Linear : DRG $\propto \log_2 M$ (10 bits) Zero : DRG $\propto \log_2(M/2)$ (6 bits)	Lower: RMS $\propto$ M (2.4722×10^{-5}) Upper: RMS $\propto$ M (4.6814×10^{-5})	Linear : CP $\propto$ N (24τ) Zero : CP $\propto$ N (23τ)
Architecture II Symmetric Functions	Linear : DRG $\propto \log_2 M$ (10 bits) Zero : DRG $\propto \log_2(M/2)$ (6 bits)	Lower: RMS $\propto$ M (2.4722×10^{-5}) Upper: RMS $\propto$ M (4.6814×10^{-5})	Linear : CP $\propto \log_2(M/2)$ (9τ) Zero : CP $\propto \log_2(M/2)$ (8τ)
Architecture III General Functions	Linear : DRG $\propto M\log_2(M/2)$ (85 bits) Zero : DRG $\propto$ M (29 bits)	Lower: RMS $\propto$ M (2.7970×10^{-4}) Upper: RMS $\propto$ M (8.1202×10^{-4})	Linear : CP $\propto$ M (48τ) Zero : CP $\propto$ M (24τ)
Architecture IV General Functions	Zero : DRG $\propto$ M (18 bits)	Lower: RMS $\propto \sqrt{M}$ (1.0790×10^{-5}) Upper: RMS $\propto$ M (6.2914×10^{-5})	Zero : CP $\propto M^2$ (62τ)
Architecture V General Functions	Zero : DRG $\propto$ M (15 bits)	Lower: RMS $\propto \sqrt{M}$ (1.0790×10^{-5}) Upper: RMS $\propto$ M (6.2914×10^{-5})	Zero : CP $\propto$ M (9τ)

Table 4.1 Comparison between the proposed Architectures

Regarding architectures IV and V, which are suitable for zero order implementation only, the values for DRG and QN are comparable. On the other hand, in terms of hardware requirements and CP, architecture V is to be prefered. However, architecture IV enjoys a simpler, modular structure as well as a structure with less cross communication. We observe from table 4.1 that architecture I is the only one that has a CP which depends on the input transform size, N. Architecture II has the shortest CP, being proportional to $\log_2(M/2)$. Although the CP of both architectures III and V is proportional to M, the proportionality constant for architecture III is larger which makes its CP larger. Architecture IV has the longest CP, which is proportional to M^2.

A comparison between architecture I and II in terms of IC realization now follows. According to the discussion in Appendix F on the VLSI implementation of architecture I in double metal 3μ CMOS technology, we point out that VLSI realization of architecture I is impractical for a "large size" AFT because of the large silicon area required by the parallel load shift register. For architecture II the parallel load operation is not required and only an N/2-stage shift register is needed to hold the input data. However, the N/2-stage shift register is required to accept data from both ends. Refering to Fig 4.5, only the left hand side shift register will remain, and the input data is fed alternately through the top (the first N/2 samples) and the bottom (the second half of the samples) of the shift register. It is possible to accommodate a shift register for architecture II which can compute three times the number of Fourier coefficients that architecture I can per IC package (the IC realization of the arithmetic components will be approximately the same for both architectures). Thus, we can conclude that architecture II is more suitable for VLSI implementation.

	DRG	QN	CP
Radix-2 FFT	B $\log_2 M$	Lower: $\sqrt{\log_2 M}$ Upper: $\sqrt{M}$	$\log_2 M$

Table 4.2 Functional dependence for the radix-2 FFT algorithm.

We now compare the AFT implementations with the implementation of a radix-2 FFT algorithm. Table 4.2 shows the functional dependence of the maximum DRG, the ratio of the rms of the error to the rms of the completed transform (QN) [26], and the critical path (CP) as a function of the transform size, M, for the decimation-in-time radix-2 FFT algorithm. We assumed in table 4.2 that the butterflies in each stage are computed in parallel, and that the real multiplications are also performed in parallel.

In table 4.2, B is the number of bits required to represent the real values of the twiddle factor, and is determined by the coefficient accuracy desired. A typical value for B is 16. Thus, it is apparent that the radix-2 FFT has a much higher dynamic range growth than the AFT. Furthermore, it is important to realize that in order to make a fair comparison between the AFT and the FFT in terms of the quantization noise, the root-mean-square of the error should be considered instead of the ratio (QN). This is so because in the AFT, we assumed the rms of the completed transform to have a magnitude of unity (since it is difficult to derive a formula for it). For the radix-2 FFT, the rms of the completed transform is derived [26] to be MK/2, where K is the average modulus squared of the initial sequence. If the rms of the error is the criterion, then the approximate upper and lower bounds on the error for the FFT increase as, respectively, M and $\sqrt{M \log_2 M}$. These results are

comparable to the quantization noise of the AFT when zero order interpolation is used. In terms of the critical path, for addition the FFT performance is similar to that of architecture II but better than the rest of the architectures for the implementing the AFT. For multiplication, the CP for architectures I, II, and V is constant (equal to one unit operation time for multiplication by a linear interpolant plus one unit operation time for division by a scalar) and consequently these architectures will be faster than the FFT. For architectures III and IV, the CP is larger than that of the FFT for both addition and multiplication. It is assumed all along that the multiplication by a twiddle factor in the FFT takes approximately the same unit of time as the division by a scalar in the AFT. This assumption is valid as long as we are considering the functional dependence (order of magnitude) of the CP and not the absolute value, which we did in the above discussion.

4.9 SUMMARY

In this chapter we have presented hardware simulations of the proposed architectures using Extend (the results of the simulations are shown in Appendix E). The hardware requirements of each architecture is discussed. We have analyzed some of the effects of using finite-precision arithmetic in the implementation of the AFT. We considered dynamic range growth(DRG) and fixed-point implementations of the AFT. A basic observation is that quantization effects depend to a great extent upon the form or structure chosen to the implement the algorithm. We have estimated the critical path or the longest computational path for the proposed architectures. We have compared the proposed architectures in terms of hardware requirements, dynamic range growth (DRG), quantization noise (QN) due to finite wordlength computation, and critical path (CP). We also compared the AFT

implementations with the implementation of a radix-2 FFT algorithm. A conclusion is that the AFT offers computation of the Fourier transform with a much lower dynamic range growth, and consequently requires smaller register size for representation of data.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 CONCLUSIONS

The main objective of this thesis has been to investigate the implementation of the recently discovered AFT algorithm in the VLSI medium. The main contributions of this work lie in the following areas: (1) studying the computational and implementational aspects of the AFT algorithm; (2) proposing several new different VLSI architectures for implementation of the AFT; (3) verifying the correctness of these proposed architectures using Extend; (4) deriving relationships between dynamic range growth (DRG), quantization noise (QN), and critical path (CP) and both input and output transform sizes for the proposed architectures.

On computing the AFT, the following conclusions are drawn. First, while the FFT requires M points of input data, uniformly distributed over the unit interval, for an M-point transform, the AFT requires on the order of M^2 points of input data, not uniformly distributed over the unit interval, to compute M Fourier coefficients. If uniform sampling is required, then for the AFT the sampling rate must have a frequency equal to 4 times the least common multiple of the integers 1,2,3,....,M. Second, if the sampling rate is

fixed at some value greater than the Nyquist rate, then, for any M, the Fourier coefficients computed by the FFT will be more accurate than those computed by the AFT. Third, if a large amount of data is available, then the AFT computation will be as accurate as and more efficient than the FFT.

Several new architectures suitable for VLSI implementation of the AFT algorithm have been proposed. The proposed architectures have been shown to offer an alternative and novel approach to high speed Fourier analysis. The first architecture [18] computes the Fourier coefficients of periodic symmetric signals and is suitable for implementation using both zero order and linear interpolation procedures. It is based on a symmetry associated with the Hermitiam symmetry of the roots of unity. The hardware requirements of the architecture is strongly dependent on the size of the input and output data as well as on the type of interpolation procedure used. The structure is not regular and is dominated by data manipulation and transfer. The architecture appears to offer an efficient hardware realization with very low dynamic range growth. Architecture II is a modification of architecture I and is based on the property that the Riemann sum can be split into two parts each of which is concerned with the samples of one half period. One of the advantages of the modified architecture is the ability to produce Fourier coefficients at every consecutive half period, assuming the clock rate is equal to the sampling rate. The second advantage is elimination of data transfer and manipulation and hence speed is improved. The third advantage is that the number of hardware components is reduced which leads to smaller silicon area in a VLSI realization.

Architecture III contains several different geometries for implementation of the AFT using zero order and linear interpolation procedures. The architecture computes the Fourier coefficients of any complex-valued periodic

signal. The computation is based on expressing the AFT in terms of a dot product between a matrix and a vector. The hardware requirements of the architecture is basically independent of the size of the input and output data as well as the interpolation procedure used; the only modification being to modify the entries of the PROMs and the periodicities of the internal control signals to account for the input and output data sizes. Architecture III provides an implementation with the least number of hardware components. One advantage of this architecture is that it is the most versatile among the proposed architectures since, with a slight modification, the same structure can be used for computation using either of the interpolation procedures as well as of any size AFT. The architecture offers the advantage of computing Fourier coefficients in parallel. The structure is regular, but with dynamic range growth higher than the rest of the proposed architectures.

Architecture IV computes the Fourier coefficients of any complex-valued periodic signal using zero order interpolation. The approach is based on repeated addressing and accumulating using memory structures. The hardware requirements are dependent on the size of the input and output data. The structure is highly regular and modular, which in turn leads to efficient IC realization. Based on zero order interpolation, the dynamic range growth appears to be midway between that of architectures I & II and that of architecture III. Architecture V, likewise, computes the Fourier coefficients of any complex-valued periodic signal using zero order interpolation. The computation is based on symmetries associated with calculation of the Riemann sums. The hardware requirements is independent of the size of the input data; the only modification being to modify the internal clocks to account for the appropriate samples to be used for the computation. The

structure is semi-regular with the dynamic range growth being similar to that of architecture IV.

It is important to note that although most of the proposed structures are not regular, the computation of Fourier coefficients is very efficient compared to standard techniques (e.g., FFT). The number of multiplications (or divisions) is relatively small especially when zero order interpolation is used. The AFT offers computation with a much lower dynamic range growth and lower root-mean-square error, especially when zero order interpolation is utilized (this conclusion is supported by the theoretical analysis conducted in chapter 4; however, this claim has not been verified experimentally).

In terms of arithmetic operations, the AFT involves O(M) multiplications (real) and $O(M^2)$ additions (complex) for a complex input signal. The radix-2 FFT [22] involves $O(M\log_2 M)$ for both multiplications (complex) and additions (complex). Thus it is apparent that the AFT algorithm involves fewer multiplications and consequently more favorable than the radix-2 FFT since multiplication is time-consuming and requires more silicon area than an addition.

During the research of this work, the following obsevations are made. One problem with implementing the AFT is the complicated indexing required in terms of grouping the appropriate samples to form the different Riemann sums. For instance, as we have seen in most of the proposed implementations, certain arrangements of the samples are required in order to perform the necessary average calculations, and a certain grouping of these averages are in turn required to generate the Fourier coefficients.

Another problem with implementing the AFT is the lack of structural regularity, which stems from the fact that different sizes of Riemann sums are

required to compute the Fourier coefficients. Another contribution to the structural irregularity of the AFT is that each coefficient depends on a different sized subset of the averages. For the case of implementing the AFT for general functions, yet another contribution to the structural irregularity of the algorithm is that the computational load for computing the imaginary component of the Fourier transform is greater than that for computing the real component of the Fourier transform, and thus makes the computational scheme unbalanced. From a VLSI design consideration prospective, an implementation of the AFT must be regular in order to maintain (area x time) product minimized, since, clearly, a nonregular structure of the implementation will require larger silicon area and longer design time.

A third problem with implementing the AFT is routing. Routing is still a serious problem in VLSI design, even with the advent of multi-level metal layers. Local and global interconnections in a VLSI chip require silicon area, and as the interconnections become denser in an algorithm implementation, the required area increases. This also in turn increases design time in custom VLSI.

5.2 RECOMMENDATIONS

This work presents an initial effort into the investigation of VLSI implementation of the AFT algorithm. The architectures and analysis contributed by this research provide a groundwork for futue developments and improvements in this domain. Some of these developments may be:

(1) To pipeline the proposed architectures.

(2) To fabricate and test some of these proposed architectures.

(3) To make a comparison between the AFT and the FFT in terms of physical realization in IC.

(4) To experimentally confirm the correctness of the relationships derived for the upper and lower bounds on the root-mean-square error.

REFERENCES

[1] D. W. Tufts and G. Sadasiv, 1988, "The Arithmetic Fourier Transform", IEEE ASSP Magazine, pp. 13-17

[2] J. W. Cooley and J. W. Tukey, 1965, "An Algorithm for the Machine calculation of Complex Fourier Series", Math. Computation, Vol. 19, pp. 297-301

[3] E. Dubois and A. N. Venetsanopoulos, 1978, "A new algorithm for the radix-3 FFT", IEEE Trans. ASSP-26, pp. 222-225

[4] S. Prakash and V. V. Rao, 1981, "A new radix-6 FFT algorithm", IEEE Trans. ASSP-29, pp. 939-941

[5] Y. Suzuki, T. Sone, and K. Kido, 1986, "A new FFT algorithm of radix 3, 6, and 12", IEEE Trans. ASSP-34, pp. 300-383

[6] C. S. Burrus and P. W. Eschenbacher, 1981, "An in-place, in-order prime factor FFT algorithm", IEEE Trans. ASSP-29, pp. 806-817

[7] D. P. Kolba and I. W. Parks, 1977, "A prime factor FFT algorithm using high-speed convolution", IEEE Trans. ASSP-25, pp. 281-294

[8] R. Duhamel and H. Hollmann, 1984, "Split-radix FFT algorithm", Electron. Letters 20, pp. 14-16

[9] Shuni Chu and C. Sidney Burrus, 1982, "A prime factor FFT algorithm using Ddistributed arithmetic", IEEE Trans. ASSP-30, pp. 217-227

[10] Ben-Dau Tseng, G. A. Jullien, and W. C. Miller, 1979, "Implementation of FFT structures using the Residue Number System", IEEE Trans. on Computers, Vol. C-28, No. 11, pp. 831-844

[11] P. Harzer, 1886, "Über eine von Herrn Tschebyschef angegebene Interpolationsmethode", Astronomische Nachrichten, pp. 115

[12] H. Bruns, 1903, "Grundlinien des wissenschaftlichen Rechnens", B.G. Teubner Verlag, Leipzig

[13] A. Wintner, 1945, "An Arithmetical Approach to Ordinary Fourier Series", Waverly, Baltimore

[14] D. W. Tufts and G. Sadasiv, 1988, "Arithmetic Fourier Transform and Adaptive Delta Modulation: a Symbiosis for High Speed Computation", SPIE, Vol. 880, pp.168-178

[15] I. S. Reed, D. W. Tufts, X. Yu, T. K. Truong, M.-T. Shih and X. Yin, "Fourier Analysis and Signal Processing by Use of the Mobius Inversion Formula", IEEE Trans. on ASSP, V. 38, pp. 458-470

[16] N. Kouvaras, 1978, "Operations on delta-modulated signals and their application in the realization of digital filters", The Radio and Electronic Engineer, Vol. 48, No. 9, pp.431-438

[17] M. T. Shih, I. S. Reed, T. K. Truong, E. Hendon, and D. W. Tufts, 1990, "A VLSI Architecture for Simplified Arithmetic Fourier Transform Algorithm",

[18] N. Wigley and G. A. Jullien, "On Implementing the Arithmetic Fourier Transform", in press IEEE Trans. on ASSP

[19] N. Wigley and G. A. Jullien, J. Benzreba, and D. Reaume, "On Computing the Arithmetic Fourier Transform", submitted to Spectral Techniques: Theory and Applications, C. Moraga and R. Creutzburg (Eds.)

[20] I. Niven and H. S. Zuckerman, 1972, "An Introduction to the Theory of Numbers", John Wiley, New York

[21] M. R. Schroeder, 1986, "Number Theory in Science and Communication", Springer-Verlag, Berlin

[22] Lawrence R. Rabinar and Bernard Gold, 1975, "Theory and Application of Digital Signal Processing", Prentice-Hall, pp. 587-594

[23] H. T. Kung, Bob Sproull and Guy Steele, 1981, "VLSI systems and Computations", Computer Science Press, pp. 278-281

[24] Carver Mead and Lynn Conway, 1980, "Algorithms for VLSI Processor Arrays" in *Introduction to VLSI systems*, Addison-Wesley series, pp.271-290

[25] Kai Hwang, 1979, "Computer arithmetic: principles, architecture, and design", NY: Wiley, pp.201-206

[26] H. H. Guild, 1970, "Some Cellular Logic Araays for Nonrestoring Binary Division", The Radio and Elec. Engr., Vol. 39, pp. 345-348

[27] R. Stefanelli, 1972, "A Suggestion for High-Speed Parallel Bianry Divider", IEEE Trans. Comp., Vol. C-21, pp. 42-55

[28] M. Cappa and V. C. Hamacher, 1973, "An Augmented Iterative Array for High-Speed Binary Division", IEEE Trans. on Computers, Vol. C-22, pp. 172-175

[29] Peter D. Welch, 1969, "A Fixed-Point Fast Fourier Transform Error Analysis", IEEE Trans. on Audio and ElectroAcoustics, Vol. AU-17, No.2, pp. 153-157

[30] Alan V. Oppenheim and Clifford J. Weinstein, 1972, "Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform", Pro. IEEE, Vol. 60, No.8, pp. 957-976

[31] A. V. Oppenheim and R. W. Schafer, 1975, "Digital Signal Processing", Prentice-Hall, pp. 447-456

[32] Douglas F. Elliott, 1975, "Handbook of Digital Signal Processing, Engineering Applications", Academic Press, pp. 591-594

[33] Jiren Yuan and Christer Svensson, 1989, "High-speed CMOS Circuit Technique", IEEE Journal of solid-state circuits, Vol. 24, No. 1

BIBLIOGRAPHY

1. N. Tepedelenliglu, 1989, "A Note on the Computational Complexity of the Arithmetic Fourier Transform", IEEE Trans. on ASSP, Vol. 37, No.7, pp. 1146-1147
2. D. W. Tufts, 1989, "Comments on "A Note on the Computational Complexity of the Arithmetic Fourier Transform"," IEEE Trans. on ASSP, Vol. 37, No.7, pp. 1147-1148
3. N. S. Jayant and P. Noll, 1984, "Digital Coding of Waveforms Principals and Applications", Prentice-Hall, Inc. Englewood, NJ
4. Gordon B. Lockhart, 1972, "Digital encoding and filtering using delta modulation", The Radio and Electronic Engineer, Vol. 42, No. 12, pp. 547-551
5. G. L. Baldwin and S. K. Tewksbury, 1974, "Linear delta modulator integrated circuit with 17-Mbit/s sampling rate", IEEE Trans. Commun., Vol. COM-22, pp. 977-985
6. R. Steele, 1975, "Delta Modulation Systems", Pentech Press, London
7. M. Abramowitz and I. A. Stegun, 1968, "Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables", National Bureau of Standards

APPENDIX A

FORTRAN CODE FOR COMPUTING THE AFT

```
C     ***************************************************************
C     PROGRAM NAME: GENERALAFT
C
C     THIS PROGRAM COMPUTES THE ARITHMETIC FOURIER
C     TRANSFORM (USING ZERO ORDER AND LINEAR
C     INTERPOLATION) AND THE DISCRETE FOURIER TRANSFORM
C     OF AN INPUT SIGNAL OF ANY LENGTH AND PRODUCES AN
C     OUTPUT SIGNAL IN ANY DESIRABLE LENGTH. THE FOURIER
C     COEFFICIENTS OF THE SIGNAL ARE CONTAINED IN AN OUTPUT
C     DATA FILE NAMED AFTOUT.
C
C     ***************************************************************
C     ***************************************************************
C     MAIN PROGRAM
C     NN IS THE LENGTH OF THE INPUT VECTOR & MM IS THE
C     LENGTH OF THE OUTPUT VECTOR. NN AND MM NEED NOT BE
C     EQUAL.
C     ***************************************************************

      DIMENSION F(0:1024),AFTZE(1024),AFTZO(1024),AFTLE(1024),
     +AFTLO(1024),DFTE(0:1024),DFTO(0:1024)
      REAL SUM,UPPER,LOWER,TOL,SUBTRACT
      EXTERNAL G
      DATA LOWER,UPPER,TOL/0.0,1.0,1.0E-5/
      OPEN(11,FILE='AFTOUT',STATUS='NEW')
      NN=60
      MM=30
      WRITE(11,123)NN,MM
      CALL MEAN(LOWER,UPPER,TOL,SUM,G)
123   FORMAT(T5,'LENGTH OF INPUT SEQUENCE=',I4,/,T5,'LENGTH OF
     +OUTPUT SEQUENCE=',I4)
      WRITE(11,122)SUM
122   FORMAT(T5,'THE MEAN IS ',F10.6,/)
      WRITE(11,6)
6     FORMAT(T3,'K',T15,'DFT',T25,'ZERO_AFT',T40,'LIN_AFT',/)
      CALL COMPUTE_SIGNAL(NN,F)
      SUBTRACT=G(0)-SUM
      CALL COMPUTE_DFT(NN,MM,DFTE,DFTO)
      CALL COMPUTE_AFT(NN,MM,F,AFTZE,AFTZO,AFTLE,AFTLO
     +,SUBTRACT)
      WRITE(11,44)
44    FORMAT(T20,'COSINE COEFFICIENTS:',/,/)
      DO 13 K=1,MM
      WRITE(11,7)K,DFTE(K),AFTZE(K),AFTLE(K)
13    CONTINUE
7     FORMAT(T2,I2,T10,F11.8,T24,F11.8,T38,F11.8,/)
```

```
      WRITE(11,21)
21    FORMAT(/,/,/,/)
      WRITE(11,55)
55    FORMAT(T20,'SINE COEFFICIENTS:',/,/)
      DO 17 J=1,MM
      WRITE(11,7)J,DFTO(J),AFTZO(J),AFTLO(J)
17    CONTINUE
      STOP
      END


C     ***********************************************************************
C     SUBPROGRAM TO COMPUTE SAMPLE VALUES
C     ***********************************************************************
      SUBROUTINE COMPUTE_SIGNAL(NP,H)
      DIMENSION H(0:NP)
      REAL T
      PI=3.1415927
      DO 5 K=1,NP
      T=(K-1)/(NP*1.0)
      H(K-1)=G(T)-G(0)
5     CONTINUE
      RETURN
      END


C     ***********************************************************************
      FUNCTION G(X)
C     ***********************************************************************
      PI=3.1415927
C     SYMMETRIC POLYNOMIAL
C     G=1-30*X**2+60*X**3-30*X**4
C     POLYNOMIAL
C     G=1-30*X+60*X**2-30*X**4
C     GAUSSIAN
C     G=EXP(-(X-0.5)**2/0.03)
C     SINUSOIDAL
C     G=SIN(2*PI*3*X)+SIN(2*PI*17*X)+SIN(2*PI*9*X)
C     LINEAR
      G=X
      RETURN
      END
```

```
C     ***********************************************************************
C     SUBPROGRAM TO CALCULATE THE MEAN OF THE INPUT
C     VECTOR
C     ***********************************************************************
      SUBROUTINE MEAN(LOWER,UPPER,TOL,SUM,G)
      REAL X,DELTA,LOWER,UPPER,SUM,TOL
      REAL ENDSUM,ODDSUM,EVSUM,SUM1
      K=2
      DELTA=(UPPER-LOWER)/K
      ODDSUM=G(LOWER+DELTA)
      EVSUM=0.0
      ENDSUM=G(LOWER)+G(UPPER)
      SUM=(ENDSUM+4*ODDSUM)*DELTA/3.0
5     K=K*2
      J=K/2
      SUM1=SUM
      DELTA=(UPPER-LOWER)/K
      EVSUM=EVSUM+ODDSUM
      ODDSUM=0.0
      DO 10 I=1,J
      X=LOWER+DELTA*(2*I-1)
      ODDSUM=ODDSUM+G(X)
10    CONTINUE
      SUM=(ENDSUM+4.0*ODDSUM+2.0*EVSUM)*DELTA/3.0
      IF(ABS(SUM-SUM1).GT.ABS(TOL*SUM).AND.K.LT.1000)GOTO 5
      RETURN
      END


C     ***********************************************************************
C     SUBPROGRAM TO COMPUTE THE DFT
C     ***********************************************************************
      SUBROUTINE COMPUTE_DFT(NP,NC,C,D)
      DIMENSION C(0:NC),D(0:NC)
      REAL PI,X
      PI=3.1415927
      DO 4 K=1,NC+1
      C(K-1)=0.0
      D(K-1)=0.0
4     CONTINUE
      DO 8 K=1,NC+1
      DO 12 J=1,NP
      X=(J-1)/(NP*1.0)
      C(K-1)=C(K-1)+G(X)*COS(2*PI*(K-1)*X)
      D(K-1)=D(K-1)+G(X)*SIN(2*PI*(K-1)*X)
12    CONTINUE
```

```
      C(K-1)=2.0*C(K-1)/(NP*1.0)
      D(K-1)=2.0*D(K-1)/(NP*1.0)
8     CONTINUE
      RETURN
      END


C     ***********************************************************************
C     SUBPROGRAM TO COMPUTE THE AFT USING ZERO ORDER AND
C     LINEAR INTERPOLATION PROCEDURES
C     ***********************************************************************
      SUBROUTINE COMPUTE_AFT(NP,NC,Q,A,B,AA,BB,SUBTRACT)
      DIMENSION Q(0:NP),A(NC),B(NC),AA(NC),BB(NC),MU(1024),
     +SZE(1024),SZO(1024),SLE(1024),SLO(1024)
      REAL FRACO,FRACE,SOTEMP,SETEMP,SOLTEMP,SELTEMP,
     +SUBTRACT
      INTEGER R,NUMER,AODD,AEVEN,XODD,XEVEN,JEVEN,JODD
      CALL CALMU(NC,MU)
      DO 3 K=1,NC
      DO 19 M=1,NC
      L=M*K
      IF(L.GT.NC)GOTO 3
      SETEMP=0.0
      SOTEMP=0.0
      SELTEMP=0.0
      SOLTEMP=0.0
      DO 9 R=1,L
      NUMER=JMOD(M+4*(R-1),4*L)
      FRACO=NUMER*1.0/(4*L)
      FRACE=(R-1)*1.0/L
      JODD=INT(NP*FRACO)
      JEVEN=INT(NP*FRACE)
      AODD=4*L*(JODD+1)-NP*NUMER
      AEVEN=L*(JEVEN+1)-NP*(R-1)
      IF(AODD.GT.(4*L-AODD))THEN
      XODD=JODD
      ELSE
      XODD=JODD+1
      ENDIF
      IF(AEVEN.GT.(L-AEVEN))THEN
      XEVEN=JEVEN
      ELSE
      XEVEN=JEVEN+1
      ENDIF
      SOTEMP=SOTEMP+Q(XODD)
      SETEMP=SETEMP+Q(XEVEN)
```

```
      SOLTEMP=SOLTEMP+((AODD*Q(JODD)+(4*L-
      +AODD)*Q(JODD+1))/(4*L))
      SELTEMP=SELTEMP+((AEVEN*Q(JEVEN)+(L-
      +AEVEN)*Q(JEVEN+1))/L)
9     CONTINUE
      SZO(L)=(SOTEMP/L)
      SZE(L)=(SETEMP/L)
      SLO(L)=(SOLTEMP/L)
      SLE(L)=(SELTEMP/L)
      A(K)=A(K)+MU(M)*(SZE(L)+SUBTRACT)
      B(K)=B(K)+MU(M)*(SZO(L)+SUBTRACT)
      AA(K)=AA(K)+MU(M)*(SLE(L)+SUBTRACT)
      BB(K)=BB(K)+MU(M)*(SLO(L)+SUBTRACT)
19    CONTINUE
3     CONTINUE
      RETURN
      END


C     ***********************************************************************
C     SUBPROGRAM TO GENERATE MU
C     ***********************************************************************
      SUBROUTINE CALMU(N,MU)
      DIMENSION IIP(1000),MU(N)
      REAL H
      INTEGER JX,S,K,COUNTER1,COUNTER2,COUNTER3,COUNTER4,
      +COUNTER5,COUNTER6,COUNTER7,COUNTER8,COUNTER9
      CALL PRIME(N,IIP,K)
      MU(1)=1
      DO 2 I=2,N
      DO 3 J=2,K
      H=(I*1.0)/(IIP(J)*IIP(J))
      H=H-INT(H)
      IF(H.NE.0)GOTO 3
      MU(I)=0
      GOTO 2
3     CONTINUE
2     CONTINUE
      DO 9 II=2,N
      COUNTER1=1
      COUNTER2=1
      COUNTER3=1
      COUNTER4=1
      COUNTER5=1
      COUNTER6=1
      COUNTER7=1
```

```
      COUNTER8=1
      COUNTER9=1
      S=0
      DO 31 IN=2,K
      JX=IIP(IN)
      S=1
      IF(II.NE.JX)GOTO 31
      MU(II)=(-1)**S
      GOTO 9
31    CONTINUE

40    DO 4 I=2,K,COUNTER1
      JX1=IIP(I)
      COUNTER2=1
50    DO 6 J=2,K,COUNTER2
      IF(I.EQ.J)GOTO 6
      JX2=JX1*IIP(J)
      S=2
      IF(II.NE.JX2)GOTO 6
      MU(II)=(-1)**S
      GOTO 9
6     CONTINUE
      COUNTER2=COUNTER2+1
      IF(COUNTER2.LE.K)GOTO 50
4     CONTINUE
      COUNTER1=COUNTER1+1
      IF(COUNTER1.LE.K)GOTO 40

300   DO 21 JM=2,K,COUNTER3
      JX3=IIP(JM)
      COUNTER4=1
200   DO 22 IM=2,K,COUNTER4
      JX4=JX3*IIP(IM)
      IF(JM.EQ.IM)GOTO 22
      COUNTER5=1
100   DO 23 KM=2,K,COUNTER5
      IF(IM.EQ.KM.OR.JM.EQ.KM)GOTO 23
      JX5=JX4*IIP(KM)
      S=3
      IF(II.NE.JX5)GOTO 23
      MU(II)=(-1)**S
      GOTO 9
23    CONTINUE
      COUNTER5=COUNTER5+1
      IF(COUNTER5.LE.K)GOTO 100
22    CONTINUE
```

```
      COUNTER4=COUNTER4+1
      IF(COUNTER4.LE.K)GOTO 200
21    CONTINUE
      COUNTER3=COUNTER3+1
      IF(COUNTER3.LE.K)GOTO 300

      IF(II.LT.210)GOTO 900
700   DO 41 IX=2,K,COUNTER6
      JX6=IIP(IX)
      COUNTER7=1
600   DO 42 IY=2,K,COUNTER7
      JX7=JX6*IIP(IY)
      IF(IX.EQ.IY)GOTO 42
      COUNTER8=1
500   DO 43 IZ=2,K,COUNTER8
      JX8=JX7*IIP(IZ)
      IF(IX.EQ.IZ.OR.IY.EQ.IZ)GOTO 43
      COUNTER9=1
400   DO 44 IW=2,K,COUNTER9
      JX9=JX8*IIP(IW)
      IF(IX.EQ.IW.OR.IY.EQ.IW.OR.IZ.EQ.IW)GOTO 44
      S=4
      IF(II.NE.JX9)GOTO 44
      MU(II)=(-1)**S
      GOTO 9
44    CONTINUE
      COUNTER9=COUNTER9+1
      IF(COUNTER9.LE.K)GOTO 400
43    CONTINUE
      COUNTER8=COUNTER8+1
      IF(COUNTER8.LE.K)GOTO 500
42    CONTINUE
      COUNTER7=COUNTER7+1
      IF(COUNTER7.LE.K)GOTO 600
41    CONTINUE
      COUNTER6=COUNTER6+1
      IF(COUNTER6.LE.K)GOTO 700
900   F=1
9     CONTINUE
      RETURN
      END
```

```
C     ***********************************************************************
C     SUBPROGRAM TO GENERATE PRIME NUMBERS FOR
C     SUBROUTINE MU
C     ***********************************************************************
      SUBROUTINE PRIME(N,IP,K)
      DIMENSION IP(N)
      REAL X
      K=1
      IP(1)=1
      DO 5 I=2,N,1
      DO 10 J=2,I,1
      X=(I*1.0)/(J*1.0)
      X=X-INT(X)
      IF(X.EQ.0.0.AND.J.NE.I)GOTO 5
10    CONTINUE
      K=K+1
      IP(K)=I
5     CONTINUE
      RETURN
      END
```

APPENDIX B

FOURIER COEFFICIENTS COMPUTED BY THE AFT METHOD FOR SELECTED FUNCTIONS

$$\text{Polynomial : } y = 1 - 30*t + 60*t^2 - 30*t^4$$

k	DFT		AFT 0th order interp.		AFT 1st order interp.	
	a_k	b_k	a_k	b_k	a_k	b_k
1	0.924E+00	-0.290E+01	0.919E+00	-0.289E+01	0.924E+00	-0.290E+01
2	0.577E-01	-0.363E+00	0.586E-01	-0.362E+00	0.577E-01	-0.361E+00
3	0.114E-01	-0.108E+00	0.126E-01	-0.118E+00	0.113E-01	-0.106E+00
4	0.361E-02	-0.453E-01	-0.140E-03	-0.450E-01	0.356E-02	-0.446E-01
5	0.148E-02	-0.232E-01	0.139E-02	-0.218E-01	0.147E-02	-0.232E-01
6	0.713E-03	-0.134E-01	0.631E-03	-0.143E-01	0.700E-03	-0.130E-01
7	0.385E-03	-0.845E-02	-0.848E-02	-0.357E-01	0.457E-03	-0.782E-02
8	0.225E-03	-0.566E-02	0.195E-02	-0.902E-02	0.234E-03	-0.547E-02
9	0.141E-03	-0.397E-02	-0.795E-03	-0.120E-01	0.159E-03	-0.328E-02
10	0.935E-04	-0.289E-02	0.926E-04	-0.255E-02	0.924E-04	-0.250E-02
11	0.628E-04	-0.216E-02	0.841E-02	0.239E-01	0.145E-03	-0.155E-02
12	0.420E-04	-0.166E-02	-0.375E-03	-0.167E-02	0.418E-04	-0.146E-02
13	0.322E-04	-0.130E-02	0.361E-02	-0.144E-01	0.644E-04	-0.126E-02
14	0.237E-04	-0.103E-02	-0.831E-03	0.836E-03	0.389E-04	-0.102E-02
15	0.205E-04	-0.830E-03	0.190E-04	-0.833E-03	0.190E-04	-0.833E-03
16	0.192E-04	-0.679E-03	0.180E-02	-0.202E-02	0.359E-04	-0.664E-03
17	0.113E-04	-0.561E-03	-0.866E-02	0.233E-01	0.939E-04	-0.738E-03
18	0.880E-05	-0.463E-03	0.910E-04	-0.104E-02	0.218E-04	-0.463E-03
19	0.821E-05	-0.384E-03	0.390E-02	0.293E-02	0.554E-04	-0.523E-03
20	0.758E-05	-0.321E-03	0.644E-05	-0.281E-03	0.644E-05	-0.209E-03
21	0.548E-05	-0.268E-03	-0.523E-03	-0.540E-02	0.364E-04	0.823E-04
22	0.400E-05	-0.224E-03	0.623E-03	0.129E-02	0.207E-04	-0.596E-06
23	0.374E-05	-0.183E-03	0.302E-02	0.833E-02	0.265E-04	0.222E-04
24	0.261E-05	-0.148E-03	0.423E-03	-0.428E-03	0.638E-05	-0.541E-04
25	-0.216E-05	-0.121E-03	0.952E-04	-0.152E-02	0.107E-04	-0.103E-03
26	0.755E-06	-0.924E-04	-0.618E-03	-0.374E-02	0.180E-04	-0.465E-05
27	0.247E-05	-0.704E-04	0.271E-03	-0.240E-02	0.154E-04	-0.595E-04
28	0.670E-05	-0.451E-04	0.250E-03	-0.139E-02	0.231E-04	-0.341E-04
29	0.160E-05	-0.240E-04	0.350E-02	-0.862E-02	0.792E-04	-0.798E-04

Table B.1: The AFT and the DFT of a 4th-order polynomial.

Linear function : y = t

k	DFT		AFT 0th order interp.		AFT 1st order interp.	
	a_k	b_k	a_k	b_k	a_k	b_k
1	-0.167E-01	-0.318E+00	0.152E-01	-0.318E+00	0.152E-01	-0.292E+00
2	-0.167E-01	-0.159E+00	-0.152E-01	-0.179E+00	-0.152E-01	-0.155E+00
3	-0.167E-01	-0.105E+00	-0.151E-01	-0.106E+00	-0.151E-01	-0.927E-01
4	-0.167E-01	-0.784E-01	0.119E-02	-0.833E-01	0.119E-02	-0.632E-01
5	-0.167E-01	-0.622E-01	-0.133E-01	-0.567E-01	-0.133E-01	-0.567E-01
6	-0.167E-01	-0.513E-01	0.278E-02	-0.583E-01	0.278E-02	-0.380E-01
7	-0.167E-01	-0.434E-01	-0.119E-01	-0.476E-01	-0.119E-01	-0.340E-01
8	-0.167E-01	-0.374E-01	-0.125E-01	-0.417E-01	-0.104E-01	-0.260E-01
9	-0.167E-01	-0.327E-01	-0.926E-02	-0.370E-01	-0.926E-02	-0.206E-01
10	-0.167E-01	-0.289E-01	-0.833E-02	-0.833E-02	-0.833E-02	-0.167E-01
11	-0.167E-01	-0.257E-01	-0.227E-01	-0.227E-01	-0.227E-01	-0.227E-01
12	-0.167E-01	-0.229E-01	-0.250E-01	-0.250E-01	-0.208E-01	-0.208E-01
13	-0.167E-01	-0.206E-01	-0.192E-01	-0.192E-01	-0.192E-01	-0.192E-01
14	-0.167E-01	-0.185E-01	-0.179E-01	-0.179E-01	-0.179E-01	-0.179E-01
15	-0.167E-01	-0.167E-01	-0.167E-01	-0.167E-01	-0.167E-01	-0.167E-01
16	-0.167E-01	-0.150E-01	-0.292E-01	-0.167E-01	-0.312E-01	-0.156E-01
17	-0.167E-01	-0.135E-01	-0.294E-01	-0.147E-01	-0.294E-01	-0.147E-01
18	-0.167E-01	-0.121E-01	-0.278E-01	-0.111E-01	-0.278E-01	-0.139E-01
19	-0.167E-01	-0.108E-01	-0.263E-01	-0.132E-01	-0.263E-01	-0.132E-01
20	-0.167E-01	-0.962E-02	-0.250E-01	-0.833E-02	-0.250E-01	-0.125E-01
21	-0.167E-01	-0.849E-02	-0.238E-01	-0.119E-01	-0.238E-01	-0.119E-01
22	-0.167E-01	-0.742E-02	-0.227E-01	-0.106E-01	-0.227E-01	-0.114E-01
23	-0.167E-01	-0.640E-02	-0.217E-01	-0.109E-01	-0.217E-01	-0.109E-01
24	-0.167E-01	-0.541E-02	-0.167E-01	-0.833E-02	-0.208E-01	-0.104E-01
25	-0.167E-01	-0.447E-02	-0.200E-01	-0.100E-01	-0.200E-01	-0.100E-01
26	-0.167E-01	-0.354E-02	-0.192E-01	-0.897E-02	-0.192E-01	-0.962E-02
27	-0.167E-01	-0.264E-02	-0.185E-01	-0.926E-02	-0.185E-01	-0.926E-02
28	-0.167E-01	-0.175E-02	-0.179E-01	-0.833E-02	-0.179E-01	-0.893E-02
29	-0.167E-01	-0.874E-03	-0.172E-01	-0.862E-02	-0.172E-01	-0.862E-02

Table B.2: The AFT and the DFT of a linear function.

Sinusoidal function : $y = \sin(6\pi^*t) + \sin(18\pi^*t) + \sin(34\pi^*t)$

k	DFT		AFT 0th order interp.		AFT 1st order interp.	
	a_k	b_k	a_k	b_k	a_k	b_k
1	0.477E-07	0.302E-06	0.309E-05	0.508E+00	0.159E-05	0.100E+00
2	0.445E-06	-0.199E-08	0.106E-05	0.273E-06	0.466E-06	0.251E-06
3	-0.676E-07	0.100E+01	0.251E-06	0.101E+01	0.843E-07	0.934E+00
4	0.401E-06	-0.346E-06	0.100E-05	0.214E-06	0.130E-05	0.265E-06
5	0.250E-06	-0.322E-06	0.326E-06	-0.442E-06	0.638E-06	-0.113E-06
6	0.290E-06	-0.592E-06	0.337E-06	-0.938E-07	0.471E-06	-0.942E-07
7	0.509E-06	-0.974E-06	-0.803E-06	0.204E+00	-0.979E-07	-0.416E-01
8	-0.783E-06	0.135E-06	-0.167E-06	0.694E-06	-0.146E-06	0.480E-06
9	-0.954E-07	0.100E+01	0.588E-07	0.110E+01	0.216E-06	0.929E+00
10	-0.362E-06	0.346E-06	0.147E-06	0.329E-06	0.153E-06	0.262E-06
11	-0.437E-06	-0.115E-06	-0.110E-05	0.258E+00	-0.387E-06	-0.408E-01
12	-0.652E-06	0.401E-06	-0.690E-06	0.199E-06	-0.723E-06	0.174E-06
13	0.789E-06	0.556E-07	0.917E-07	0.510E-01	0.871E-07	0.413E-02
14	-0.147E-05	0.699E-06	-0.664E-06	0.724E-07	-0.588E-06	0.226E-06
15	-0.239E-07	0.509E-06	-0.374E-06	0.815E-06	-0.389E-06	0.815E-06
16	-0.353E-07	0.842E-06	0.406E-06	0.261E-06	0.134E-06	0.255E-06
17	0.126E-06	0.100E+01	0.179E-05	0.105E+01	0.157E-05	0.792E+00
18	-0.463E-06	-0.811E-06	-0.107E-06	-0.839E-06	-0.256E-06	-0.760E-06
19	0.258E-07	-0.262E-06	-0.551E-06	0.438E-01	-0.287E-06	-0.192E-02
20	-0.106E-05	0.111E-06	-0.992E-06	-0.352E-07	-0.998E-06	-0.292E-06
21	0.397E-08	-0.175E-06	-0.397E-06	-0.164E-01	-0.341E-06	0.189E-02
22	-0.703E-06	0.109E-05	-0.475E-06	-0.488E-06	-0.271E-06	-0.266E-06
23	0.203E-06	-0.914E-07	-0.675E-06	0.174E+00	-0.325E-06	-0.175E-01
24	-0.636E-06	-0.572E-06	-0.414E-06	0.850E-08	-0.386E-06	-0.180E-06
25	0.104E-05	0.366E-06	0.292E-07	-0.328E-07	-0.266E-06	-0.343E-06
26	-0.203E-05	-0.594E-06	-0.837E-06	-0.465E-06	-0.511E-06	-0.264E-06
27	0.274E-06	0.116E-05	-0.512E-06	0.742E-01	-0.524E-06	-0.290E-02
28	0.302E-06	0.244E-05	0.823E-07	0.184E-06	0.525E-07	-0.412E-07
29	-0.191E-06	0.528E-06	0.159E-06	0.656E-01	-0.182E-06	0.209E-02

Table B.3: The AFT and the DFT of a sinusoidal function.

Gaussian function : $y = \exp[-(t-1/2)^2/0.03]$

k	DFT		AFT 0th order interp.		AFT 1st order interp.	
	a_k	b_k	a_k	b_k	a_k	b_k
1	-0.457E+00	-0.207E-07	-0.458E+00	0.545E-02	-0.457E+00	0.239E-03
2	0.188E+00	0.921E-08	0.191E+00	-0.136E-01	0.188E+00	-0.238E-05
3	-0.428E-01	0.137E-07	-0.474E-01	-0.715E-06	-0.425E-01	-0.298E-07
4	0.536E-02	0.311E-08	0.491E-02	0.380E-03	0.527E-02	0.271E-04
5	-0.391E-03	0.266E-07	-0.370E-03	-0.188E-05	-0.390E-03	0.179E-06
6	0.568E-06	-0.150E-07	0.566E-06	-0.301E-05	0.158E-05	0.149E-06
7	-0.119E-04	0.855E-07	0.437E-02	-0.805E-06	-0.547E-04	-0.298E-07
8	-0.990E-05	-0.147E-07	0.300E-03	-0.170E-05	0.229E-04	0.596E-07
9	-0.844E-05	0.789E-07	0.325E-02	-0.983E-06	-0.141E-03	0.328E-06
10	-0.733E-05	-0.898E-07	-0.694E-05	0.402E-05	-0.694E-05	0.298E-06
11	-0.639E-05	0.960E-07	-0.465E-02	0.000E+00	-0.539E-04	0.000E+00
12	-0.583E-05	-0.611E-08	-0.596E-05	0.000E+00	-0.596E-05	-0.298E-07
13	-0.492E-05	0.271E-06	-0.812E-03	0.000E+00	-0.460E-04	0.298E-07
14	-0.452E-05	-0.236E-06	-0.287E-02	0.280E-02	0.392E-04	0.000E+00
15	-0.437E-05	0.182E-06	-0.408E-05	-0.298E-07	-0.408E-05	-0.298E-07
16	-0.341E-05	-0.134E-07	-0.256E-03	-0.342E-03	0.213E-04	0.205E-04
17	-0.353E-05	-0.142E-06	0.257E-02	0.166E-02	-0.262E-04	-0.148E-04
18	-0.302E-05	0.860E-07	-0.283E-05	0.170E-05	-0.384E-05	0.149E-06
19	-0.300E-05	0.336E-07	-0.270E-02	-0.121E-02	-0.119E-03	-0.557E-05
20	-0.259E-05	0.179E-06	-0.316E-05	0.116E-05	-0.316E-05	0.298E-07
21	-0.272E-05	0.691E-07	0.210E-02	0.805E-06	-0.107E-03	-0.298E-07
22	-0.260E-05	-0.154E-07	0.209E-02	0.188E-02	0.211E-04	0.235E-05
23	-0.271E-05	0.480E-07	-0.295E-02	0.755E-03	-0.510E-05	-0.143E-05
24	-0.284E-05	-0.132E-06	-0.241E-05	0.170E-05	-0.241E-05	-0.894E-07
25	-0.264E-05	-0.326E-07	-0.233E-04	0.194E-05	-0.334E-05	-0.119E-06
26	-0.228E-05	0.517E-07	-0.217E-02	-0.152E-02	0.264E-04	-0.268E-06
27	-0.231E-05	-0.643E-07	-0.753E-03	0.983E-06	-0.149E-04	-0.358E-06
28	-0.245E-05	0.435E-06	0.390E-03	-0.381E-03	0.269E-04	-0.271E-04
29	-0.239E-05	-0.495E-06	0.570E-02	-0.666E-02	0.225E-03	-0.217E-03

Table B.4: The AFT and the DFT of a gaussian function.

APPENDIX C

COMPUTATIONAL COMPLEXITY OF THE AFT

The following are derived formulae for the total number of complex additions and divisions involved in computation of the exact AFT.

Exact formulae:

$$\text{Total number of additions} = \sum_{k=1}^{M} \sum_{m=1}^{[M/k]} (mk - 1)\, |\mu(m)|$$

$$\text{Total number of divisions} = (M - 1) + \sum_{i=0}^{q} \sum_{j=1}^{[M/2^i]} |\mu(j)|$$

Approximate formulae:

$$\text{Total number of additions} \approx (M^2/2) \ln M - M \ln M + (M/2)$$

$$\text{Total number of divisions} \approx (5/2)\, M - 1 \qquad \text{(J. Benzreba)}$$

$$\text{Total number of additions} = (1/2)(3M^2 + 4M\log M - 9M + 6)$$

$$\text{Total number of divisions} = 3(M - 1) \qquad \text{(Reid et al [15])}$$

The approximate formulae constitute an upper bound on the total number of operations.

The following table shows a comparison between the AFT and the radix-2 FFT (based on [22]) in terms of the total number of multiplications (divisions for the AFT). Note that in the FFT algorithm the multiplications are complex, whereas in the AFT algorithm the divisors are real integers. Therefore, it is evident from the table that for large M, the AFT will involve fewer multiplications.

Transform size, M	Approxi. (Reid [15]) (AFT)	Approxi. (J. Benzreba) (AFT)	Exact AFT	FFT Radix-2 [22]
2	3	4	3	0
4	9	9	8	2
8	21	19	18	4
16	45	39	37	24
32	93	79	73	64
64	189	159	144	160
128	381	319	286	384
256	765	639	570	896
512	1533	1279	-	2048
1024	3069	2559	-	4608

C.1 : Comparison between the AFT and the FFT
in terms of the total number of multiplications.

APPENDIX D

THE THEORY AND NOTATION FOR THE FINITE WORDLENGTH EFFECTS

Of obvious practical importance is the issue of what accuracy is to be expected when the AFT is implemented in a finite-word-length computer algorithm or special-purpose digital hardware. Among the effects of finite word length are errors due to A/D conversion of the analog signal, arithmetic roundoffs, overflow, and parameter quantization. Depending on the type of arithmetic used, fixed-point or floating-point arithmetic, one can generally estimate the performance of an algorithm based on the above mentioned effects. Here we will consider fixed-point implementation of the AFT.

In what follows, we will assume that the numbers are scaled so that the binary point lies at the extreme left. We will also assume that the input samples (i.e., the real and imaginary parts of g_k) are each represented by B bits plus a sign. The statistical model of the noise assumes the following: (1) the error is a white-noise process; (2) the probability distribution of the error is uniform over the range of quantization error; (3) the error is uncorrelated with the input signal, i.e. the error is independent of the signal, and with other errors. These assumptions appear to be valid if the signal is sufficiently complex and the quantization steps sufficiently small so that the amplitude of the signal is likely to traverse many quantization steps in going from sample to sample.

We now briefly quote the expressions [22,29-32] for the noise sources.

A/D conversion noise: Let g(n) be a sample expressed to infinite precision and $g_Q(n)$ be the sample expressed by a finite number of bits. Then, the difference signal $e(n) = g(n) - g_Q(n)$ is called quantizing noise or A/D conversion noise. We will designate the variance of this error by σ_Q^2

Overflow noise: If the result of adding two B-bit numbers causes an overflow, then the sum must be shifted right and a bit is lost. If the low-order bit shifted out is a 0, there is no error. If it is a 1, an error of 2^{-B} is incurred depending on whether the number is positive or negative. The mean and variance of this error are, respectively, zero and $\sigma_a^2 = 2^{-2B}/2$.

Roundoff noise: There are two cases to consider:

a- The first error is associated with the multiplication operation resulting from multiplying the signal values with the linear interpolants. Assuming that the linear interpolants are represented by B bits, then when the two B-bit numbers are multiplied, the result is a 2B-bit number. If the low-order B bits are rounded, then an error that is uniformly distributed between $-2^{-B}/2$ and $2^{-B}/2$, with zero mean and variance $\sigma_d^2 = 2^{-2B}/12$ is incurred.

b- The second error is associated with the division operation. We will assume that the noise injection is caused by truncating the result of division to B bits. This produces an error with mean, m_d, and variance, σ_d^2, equal to, $-2^{-B}/2$ and $2^{-2B}/12$, respectively.

APPENDIX E

EXTEND SIMULATIONS OF THE PROFOSED ARCHITECTURES

This Appendix contains the results of Extend simulations of the proposed architectures. The correctness of these results can be confirmed by comparing to the results of Appendix B for the appropriate input signal. ModL scripts of the components of each architecture as well as Extend simulations of these architectures are contained electronically in a floppy disk.

The desired values of the Fourier coefficients (read along the vertical direction) are contained at the following timesteps. For architectures I and V, the desired Foureir coefficeints are found in timestep 59. For architecture II, the desired AFT is found at timesteps 59, 89, and 119. For architecture III, the real component of the AFT is located at the consecutive timesteps starting from 731 and separated by 12 timesteps. The imaginary component of the AFT is located at the consecutive timesteps starting from 1097 and separated by 18 timesteps. For architecture IV, the real component and the imaginary componentof the AFT are located, respectively, at timesteps 137 and 179. The values at other timesteps are the instantanous values from the output nodes during the loading of the input sequence, and therefore are meaningless.

E.1 SIMULATION RESULTS OF ARCHITECTURE I

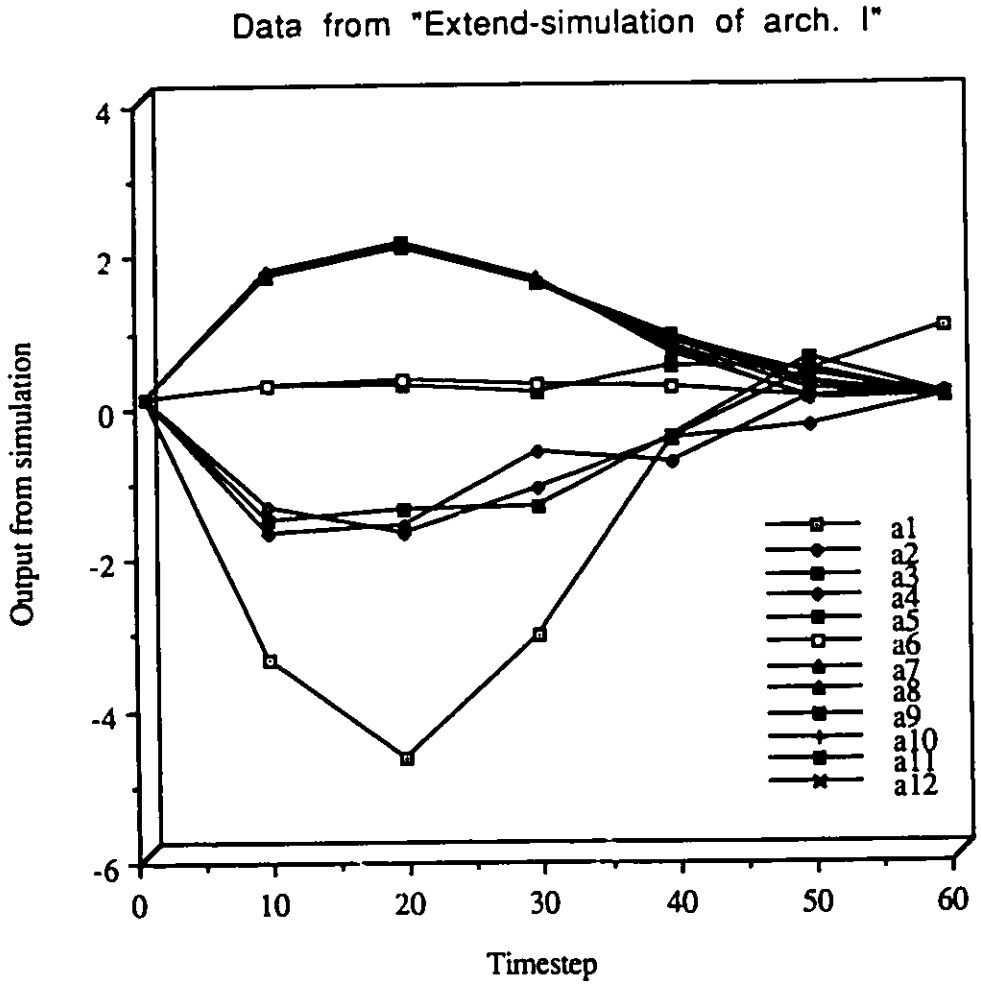


Fig E.1: Results of Extend simulation for computation of the real component of the AFT using architecture I.

E.2 SIMULATION RESULTS OF ARCHITECTURE II

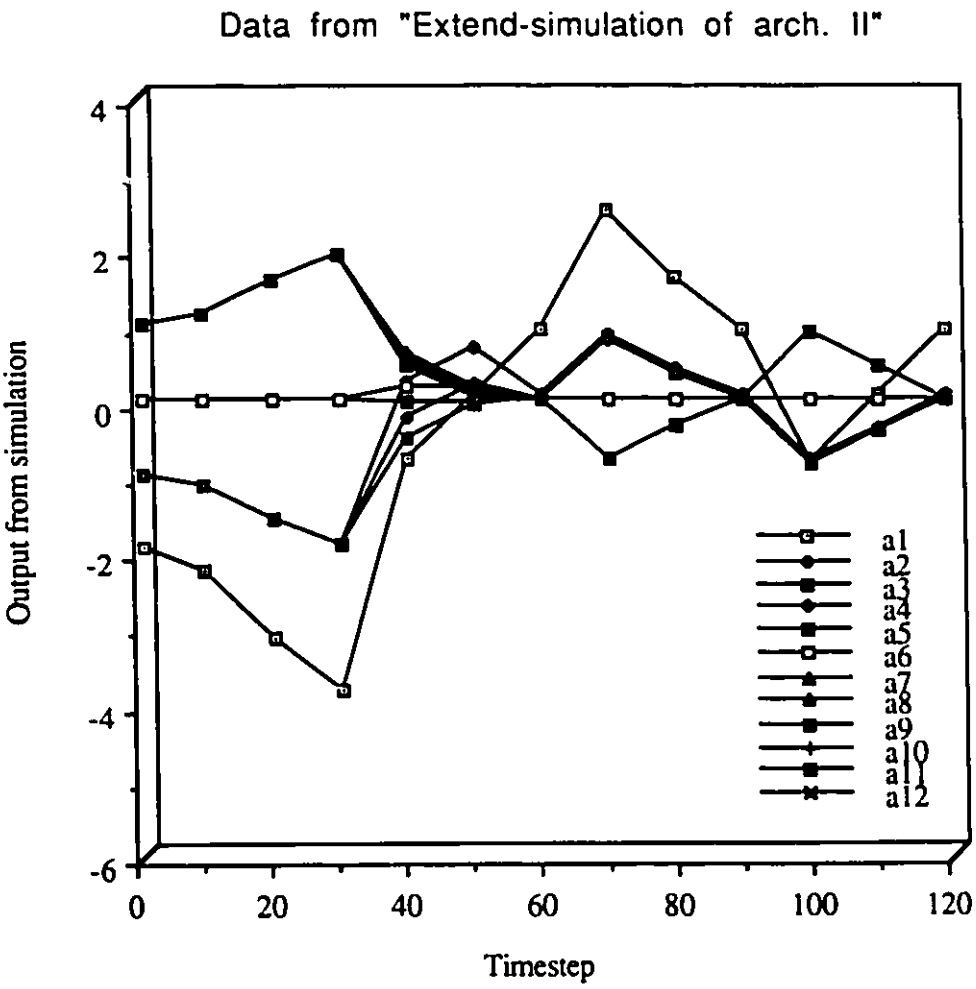


Fig E.2: Results of Extend simulation for computation of the real component of the AFT using architecture II.

E.3 SIMULATION RESULTS OF ARCHITECTURE III

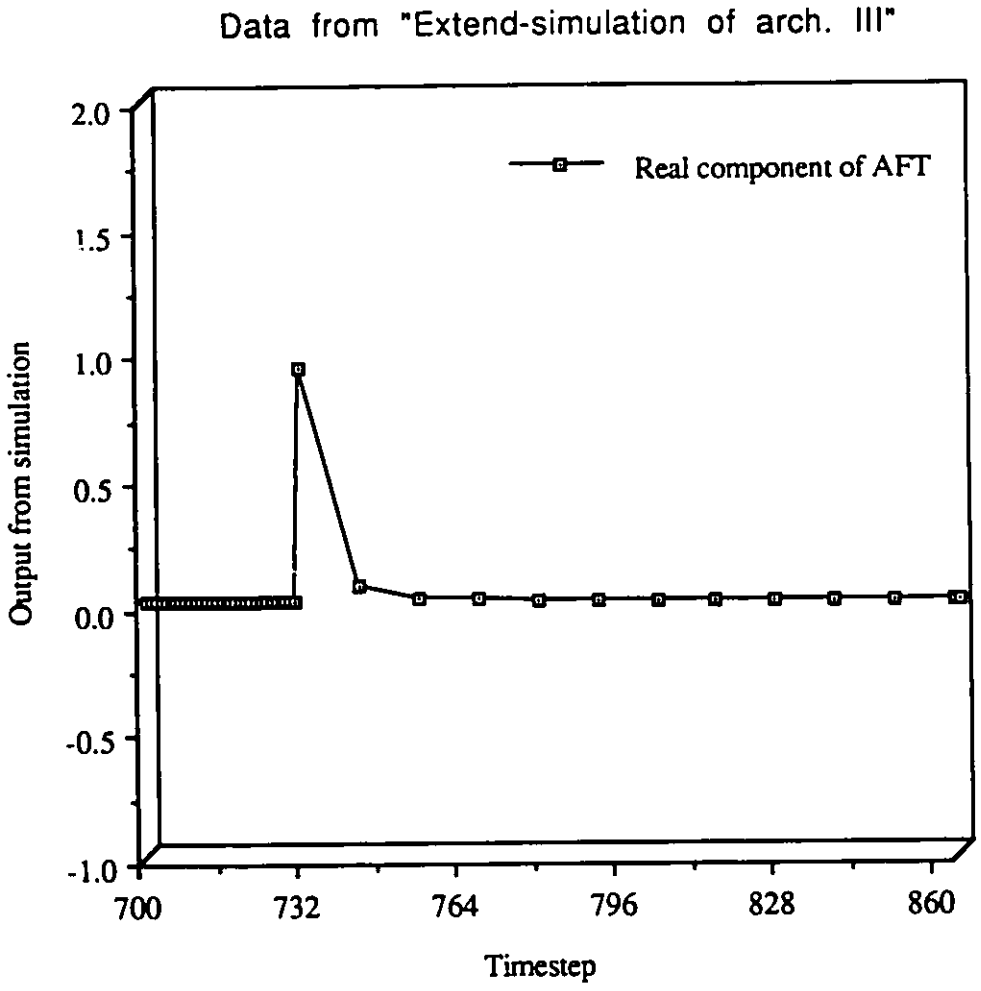


Fig E.3(a): Results of Extend simulation for computation of the real component of the AFT using architecture III.

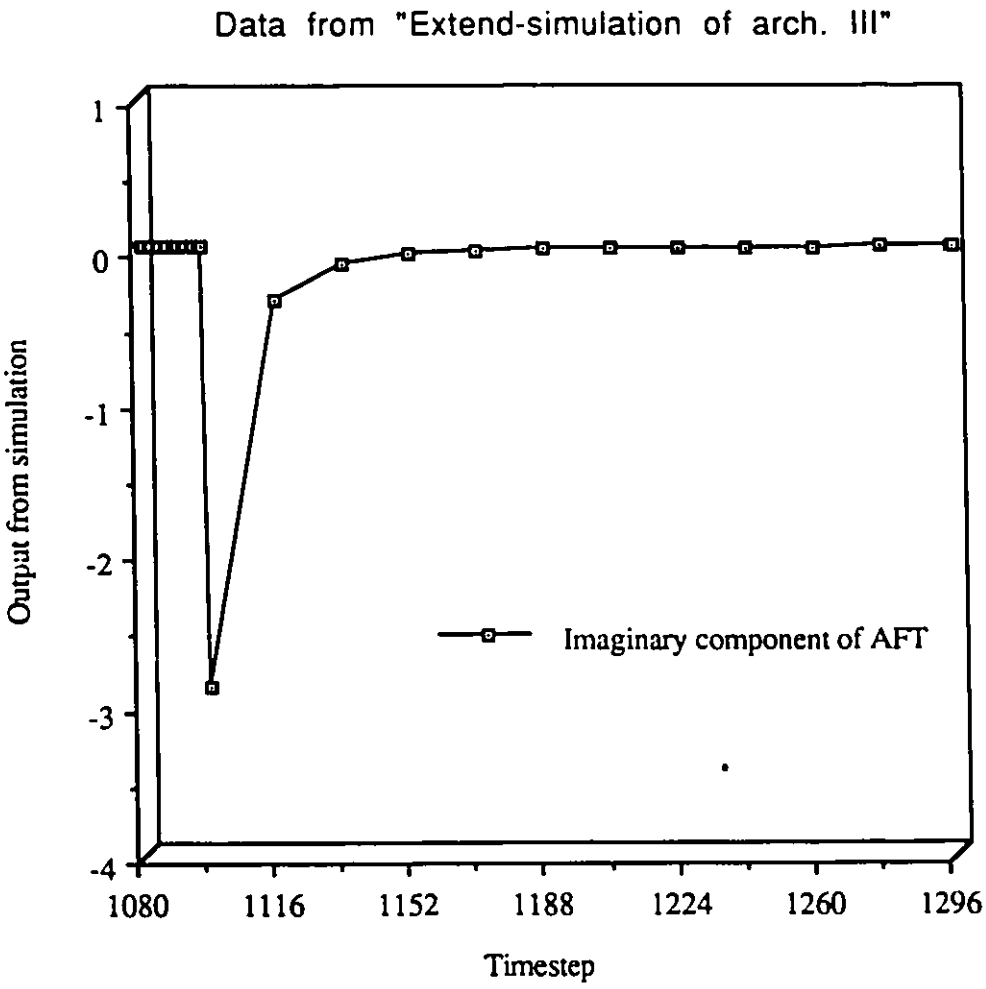


Fig E.3(b): Results of Extend simulation for computation of the imaginary component of the AFT using architecture III.

E.4 SIMULATION RESULTS OF ARCHITECTURE IV

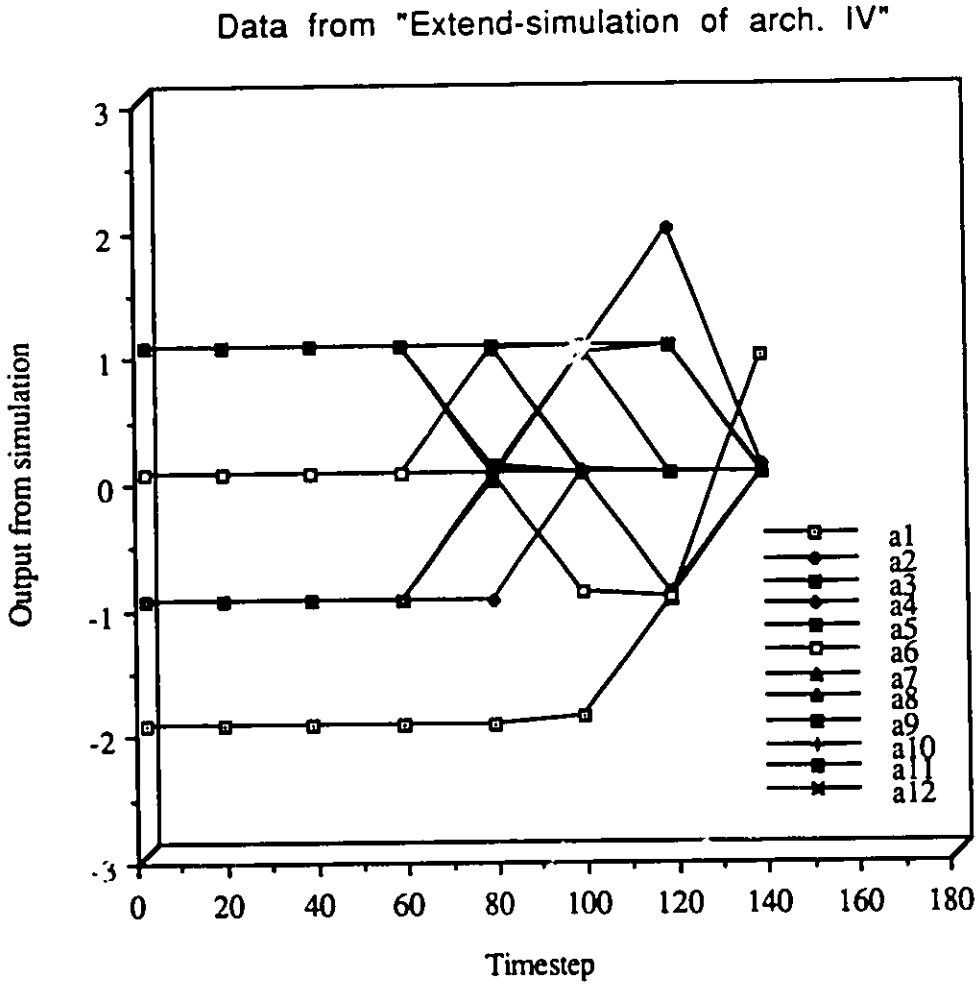


Fig E.4(a): Results of Extend simulation for computation of the real component of the AFT using architecture IV.

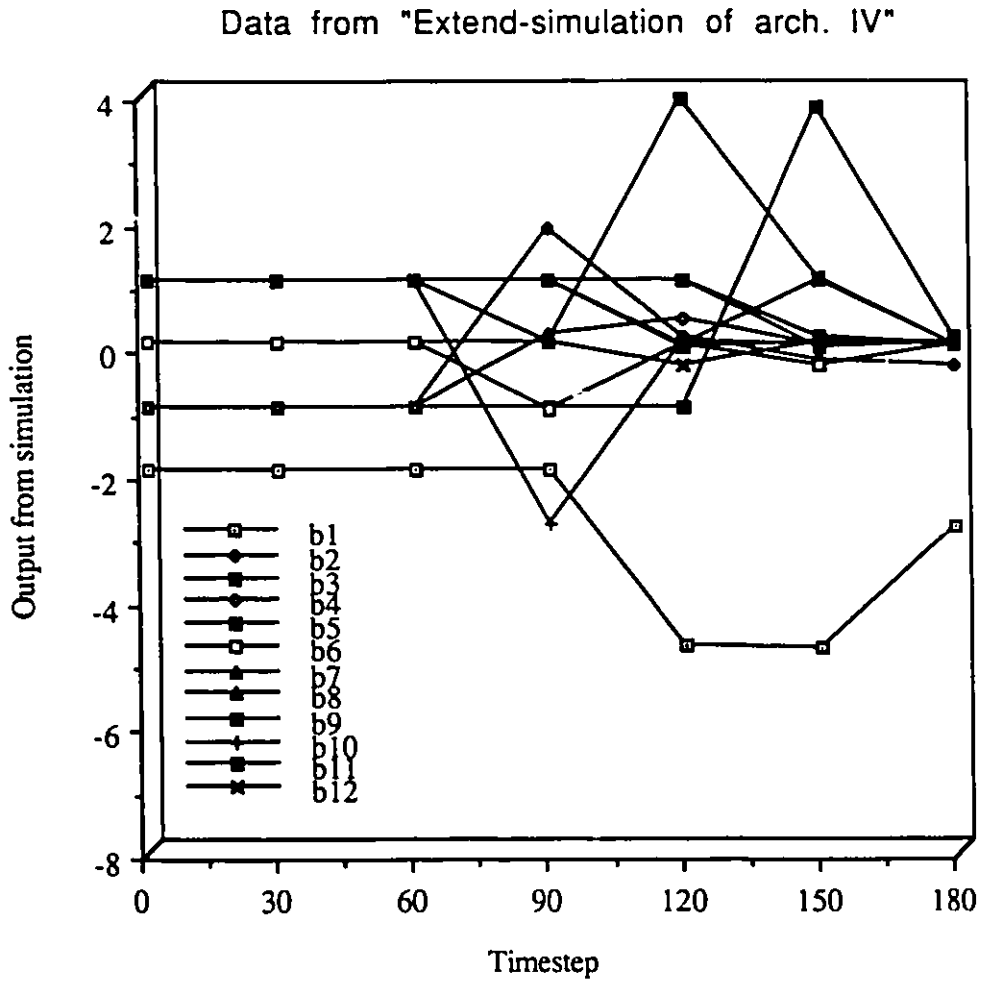


Fig E.4(b): Results of Extend simulation for computation of the imaginary component of the AFT using architecture IV.

E.5 SIMULATION RESULTS OF ARCHITECTURE V

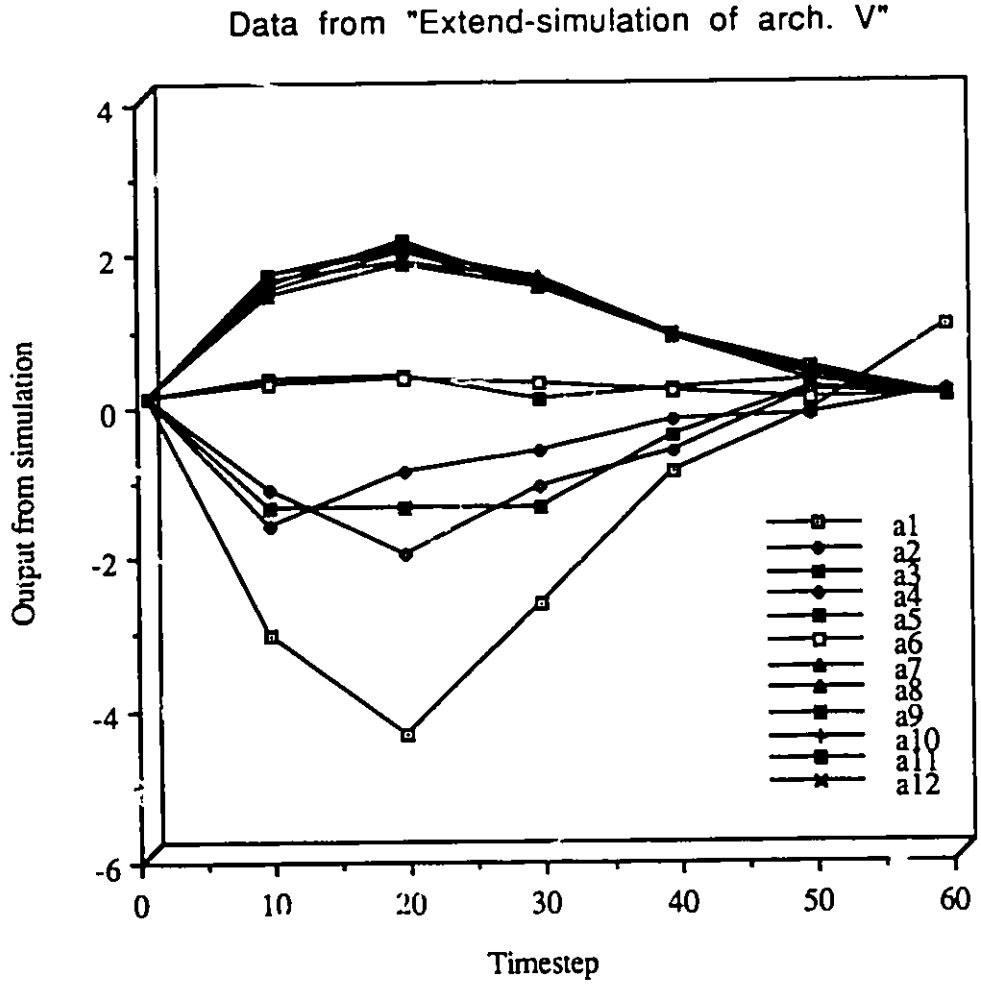


Fig E.5: Results of Extend simulation for computation of the real component of the AFT using architecture V.

APPENDIX F

VLSI IMPLEMENTATION OF THE PARALLEL LOAD SHIFT REGISTER

F.1 INTRODUCTION

In this portion of the thesis, the design and the layout of a 4-stage-8-bit parallel load shift register (PLSR) in double metal 3μ CMOS technology is undertaken. The design is based on CMOS dynamic circuits and utilizes the true single-phase clocking scheme [33]. HSPICE simulations indicated that the PLSR chip can work at clock frequencies up to the range 166-200 MHz. The fabricated chip is tested using the ASIX tools.

F.2 DESIGN CIRCUITS

The 4-stage-8-bit parallel load shift register, is part of architecture I for implementing the AFT algorithm, consists of a 4-stage-8-bit serial shift register (left hand side shift register) and a 4-stage-8-bit parallel shift register (right hand side shift register). The serial shift register transfers one bit of data every one clock pulse and the parallel shift register transfers one word (8 bits) of data every one clock pulse. The LHS and RHS shift registers are connected in such a way as to allow the movement of data, once the LHS shift register is full, from the LHS shift register to the RHS shift register in one particular clock pulse (parallel connection). Data transfer between the LHS and RHS shift registers is controlled through the use of transmission gates. A diagram of the parallel load shift register is depicted in Fig F.1.

The true single-phase clock dynamic CMOS circuit technique [33] uses one clock signal which is never inverted and consequently a high clock frequency can be reached. In this scheme, an N-latch and a P-latch are used alternatively using the same clock signal. The key in this scheme is that the N-latch and the P-latch are effectively isolated from each other during the transfer of data from the input node to the output node of the same latch.

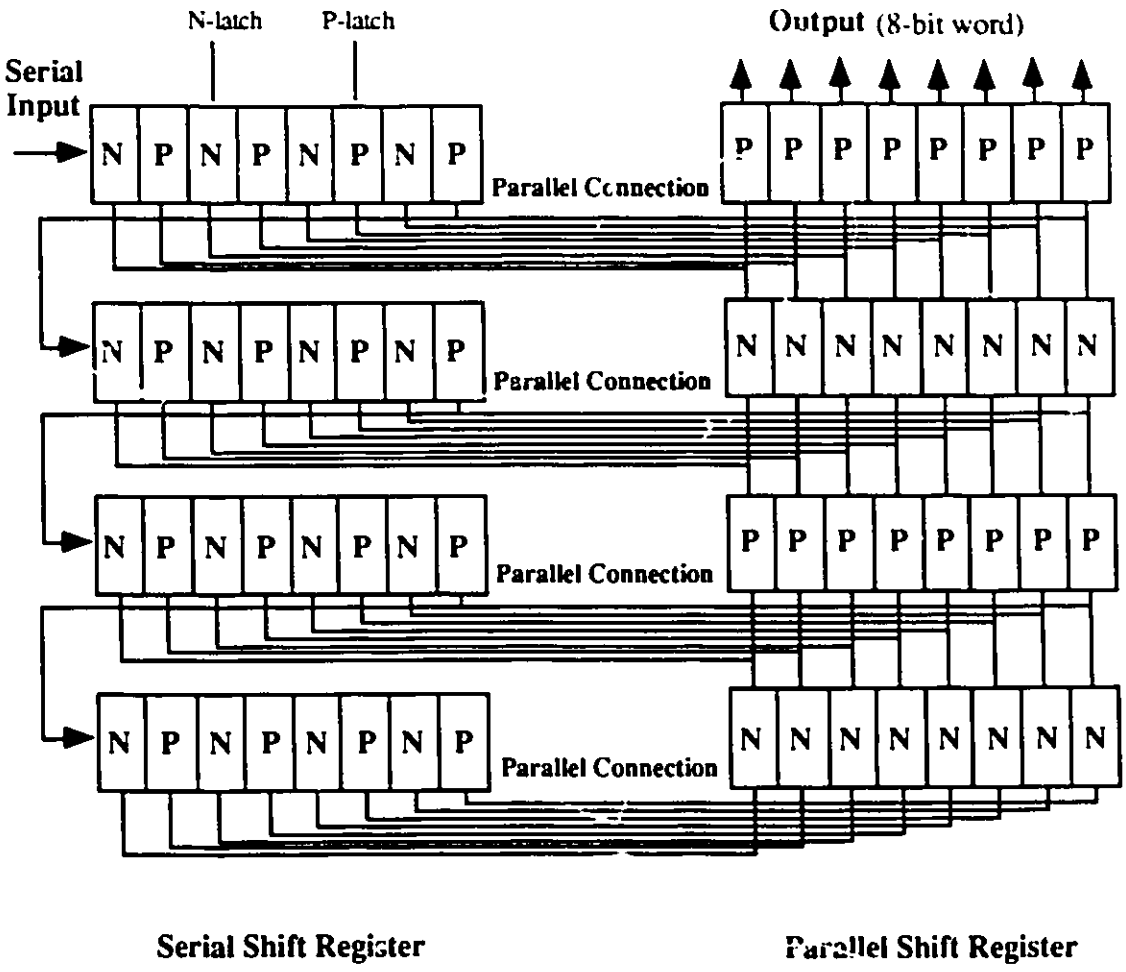
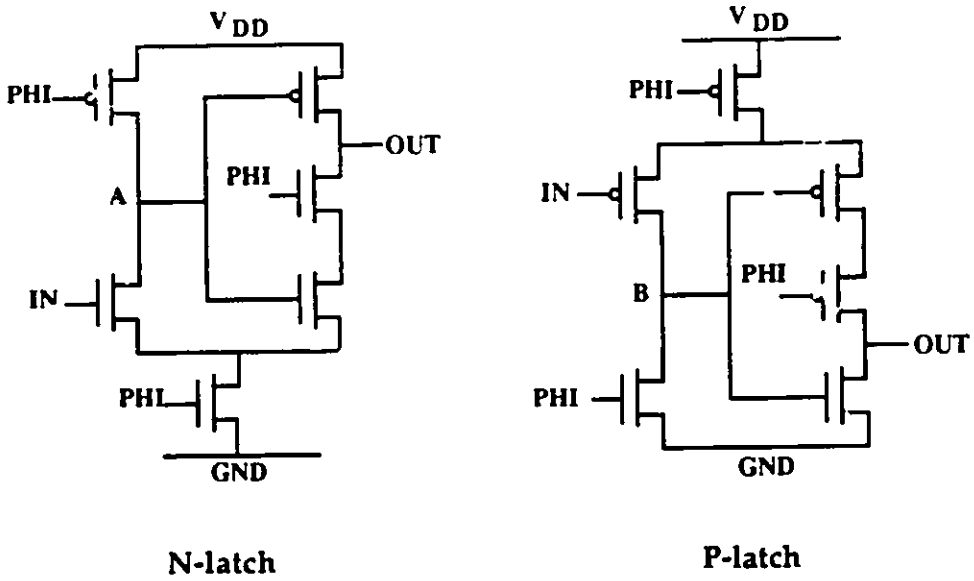


Fig F.1: Block diagram of the 4-stage-8-bit parallel load shift register.

The serial shift register is constructed by cascading N and P latches in an alternating fashion, with the movement of data controlled by a single clock signal. For the parallel shift register, N and P latches are cascaded along the bit line. Along the word line the latches are identical (either N or P type). The schematics for the N and P latches are shown in Fig F.2. Basically, each latch consists of two invertor stages controlled by a clock signal. In the diagrams, A is the precharge node for the N-latch and B is the discharge node for the P-latch.

The operation of the circuits is as follows. For the N-latch, when PHI is low,



N-latch P-latch

Fig F.2: Schematics for the N and P latches.

node A precharges to logic high and the second invertor stage is nonconducting. When PHI is high, the second invertor stage becomes conducting and depending on the input signal IN (if IN is a logic high node A discharges to logic low and if IN is a logic low node A remains charged) the logic of node A is inverted to output signal OUT and correct latching is established. For the P-latch, the reverse happens. When PHI is high, node B discharges to logic low and the second invertor stage is nonconducting. When PHI is low, the second invertor stage becomes conducting and depending on the input signal IN (if IN is a logic high node B is floating and if IN is a logic low node B charges to logic high) the logic of node B is inverted to output signal OUT.

The control process for the parallel motion of data between the LHS shift register and the RHS shift register is accomplished by using transmission gates. A transmission gate is a passive analog switch as shown in Fig F.3. A PFET and an NFET are connected in parallel, and are driven by a pair of complementary clocks, CK and CKBAR. When CK is high and CKBAR is low,

the transmission gate passes correct CMOS logic levels to the output. When CK is low and CKBAR is high, the two terminals of the transmission gate are disconnected. A single pass transistor(e.g., an NFET or a PFET) can not correctly pass both CMOS logic levels. For example, an NFET is unable to drive to a voltage higher than VDD minus the threshold voltage of the NFET, and therefore correct CMOS logic levels cannot be transmitted. In this design, a buffer stage is added to the transmission gate so that full voltage swing is obtained.

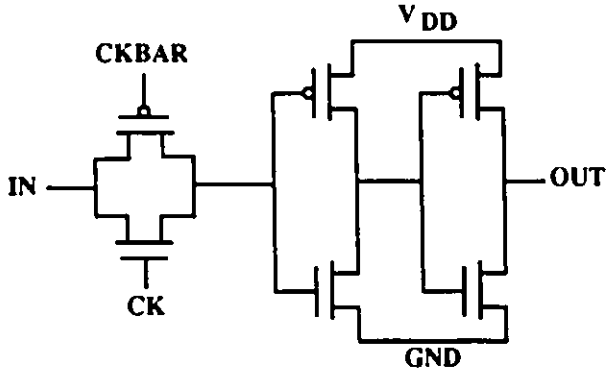


Fig F.3: Schematic for the transmission gate with buffer stage.

F.3 LAYOUT AND HSPICE SIMULATIONS

The physical layouts of the N-latch and the P-latch are shown, respectively, in Fig F.4(a) and Fig F.5(a). For the N-latch, the transistor sizes in design scale range from 25 to 50 microns for n-type transistors and 135 microns for p-type transistors. For the P-latch, the transistor sizes range from 40 to 80 for n-type transistors and 110 to 135 microns for P-type transistors. The N and P block sizes in design scale are 100 microns by 293 microns. The maximum clock frequency is found to be in the range 166-200 MHz for both N-latch and P-latch according to HSPICE simulations. Fig F.4(b) and Fig F.5(b) show the latches response to a sequence of input signals as run on HSPICE.

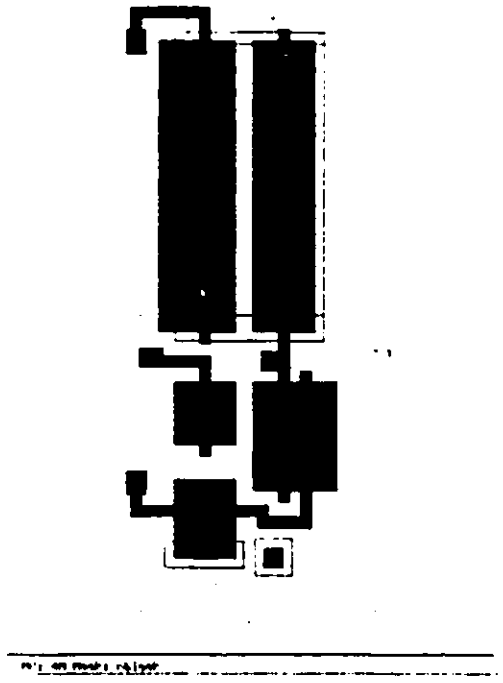


Fig F.4(a): N-latch layout.

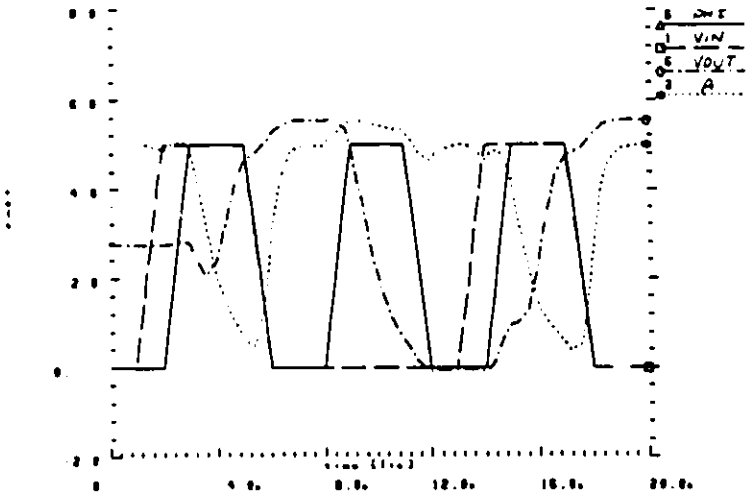


Fig F.4(b): N-latch HSPICE results.

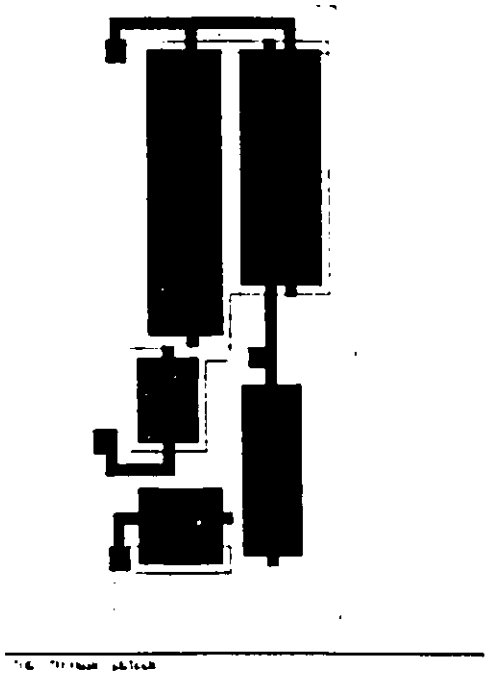


Fig F.5(a): P-latch layout.

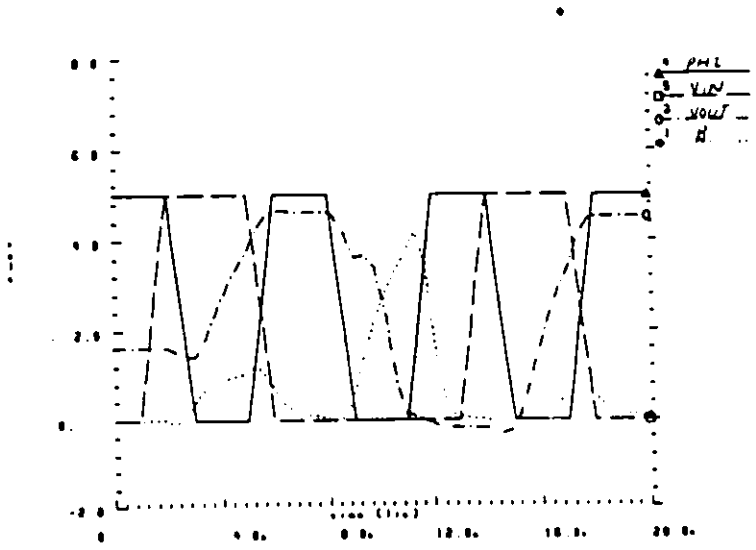


Fig F.5(b): P-latch HSPICE results.

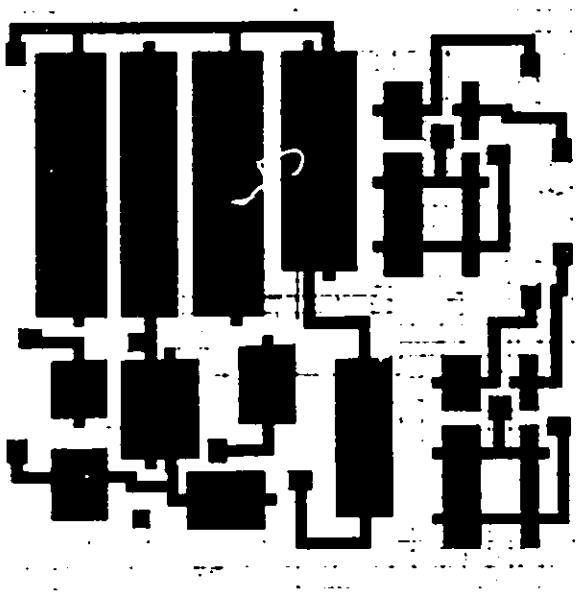


Fig F.6: NP-block layout.

The PLSR chip layout consists of three standard cells: N-block; P-block; and NP-block. The NP-block, which forms the standard cell for the LHS shift register, consists of an N-latch, P-latch, and two transmission gates with buffer stages. It was required to layout the system blocks differently in each of the LHS and RHS shift registers for the purpose of an efficient use of the chip area. The layout of the NP-block is shown in Fig F.6. The dimensions in design scale is 293 microns by 293 microns.

The overall system uses three clock signals: one clock signal for the serial shift register; one clock signal for the parallel shift register; and one clock signal and its complement for the transmission gates. HSPICE simulations were carried out and are shown in Fig F.7. It was found that the maximum working frequency is in the range 166-200 MHz. The layout of the final chip (4-stage-8-bit PLSR) is shown in Fig F.8. The dimensions in design scale is 5200 microns by 5100 microns and the number of bonding pads is 33.

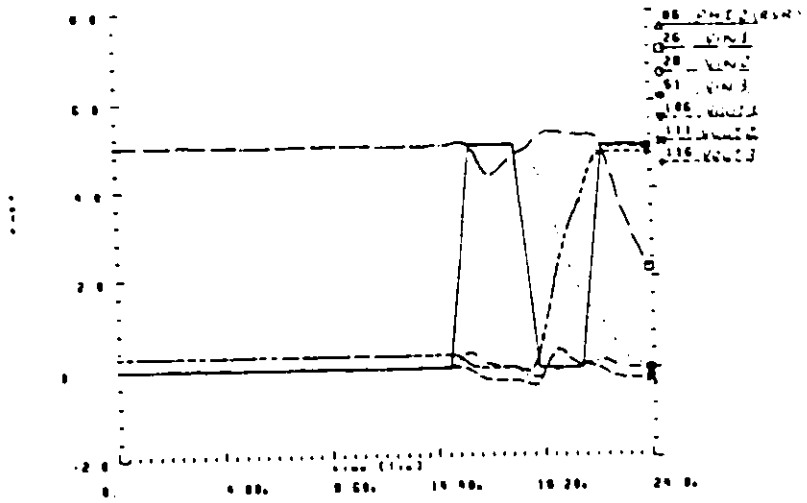


Fig F.7: PLSR HSPICE results.

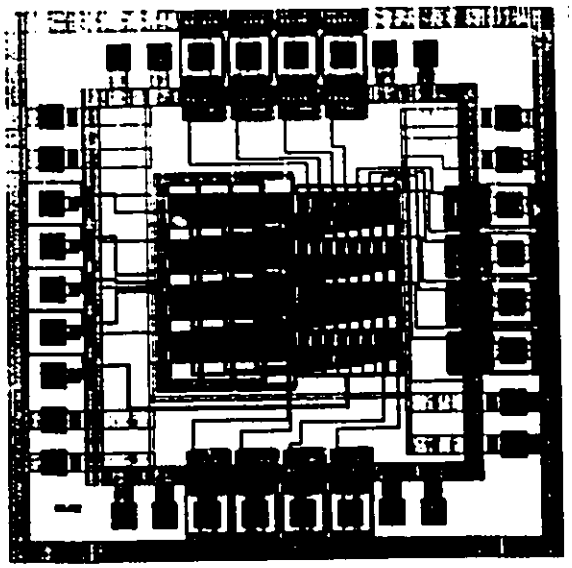


Fig F.8: The 4-stage-8-bit PLSR layout.

F.4 ASIX TESTER RESULTS

The serial shift register (SSR) is controlled by the clock signal PHI 1. After all the latches are loaded control signal (complementary clocks) CK and CKBAR are set to, respectively, high and low. The parallel shift register (PSR) is controlled by the clock signal PHI 2. The clock frequency applied to the chip is the maximum clock frequency the ASIX tester can supply in the normal mode (PHI 1 = PHI 2 = 25MHz). Fig F.9 and Fig F.10 depict test signals obtained from Tektronix 11402A digitizing oscilloscope connected to the ASIX tester. In Fig F.9 are shown the input test signal (a sequence of 1's and 0's), the clock signal PHI 1, and the response of the last latch of the SSR chain. Fig F.10 shows the control signals CK and CKBAR, the clock signal PHI 2, and the response of the output stage of the PSR (only a few waveforms are shown). Overall, the PLSR chip is found to be functional. The SSR is fully operational. However, due to difficulty in synchronizing the operation of the PSR, the chip has not been completely tested.

The relevant conclusion to be made here is that, based on this design, the VLSI realization of a 4-stage-8-bit parallel load shift register in double metal 3μ CMOS technology occupies a silicon area of size 5200μ x 5100μ in design scale. This design consumed approximately 50% of the 8mmx8mm die area of a standard 40 I/O pins package. This implies that a 8-stage-8-bit PLSR can be fitted into one such standard package (we assume that the representation of the input signal with 8 bits or one byte is reasonable). With more effort, we can probably fit a few more stages into the die area. Therefore, based on this design, the VLSI implementation of a k-stage-one-byte parallel load shift register needed in architecture I for computing the AFT of a symmetric signal can be accommodated in one IC package for $k \leq 10$, or equivalently, for computing maximum of 5 Fourier coefficients. For larger AFT sizes, the

required number of IC packages and consequently the cost of IC realization increases linearly. In addition, for k≥10 the problems of chip-to-chip communication have to be faced.

As far as the rest of the arithmetic components is concerned, the required silicon area is a function of whether the architecture is pipelined or not. In a pipelined system, we can probably reduce the arithmetic components to a high speed multiplier, an adder, and a set of registers for storing intermediate data. These will probably be fitted in one IC package (40 I/O pins), independent of the AFT size. Thus, it seems that the VLSI implementation of the parallel load shift register needed in architecture I is at least comparable to (for small size AFT: N≤10 and consequently M≤5) or larger than (for N>10) the VLSI implementation of the rest of the architecture. This feature makes the VLSI implementation of architecture I impractical for "large size" AFT.

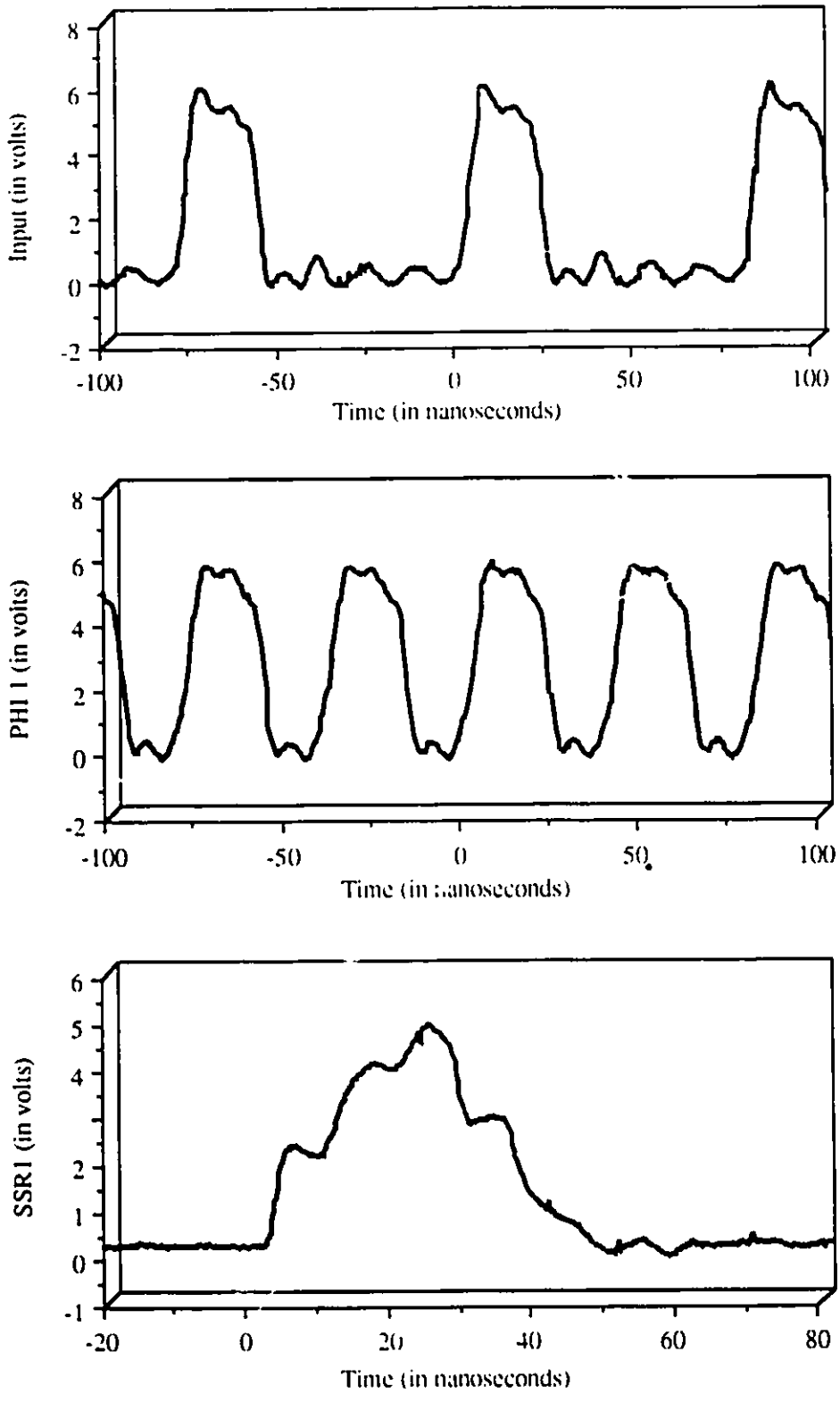


Fig F.9: SSR Waveforms.

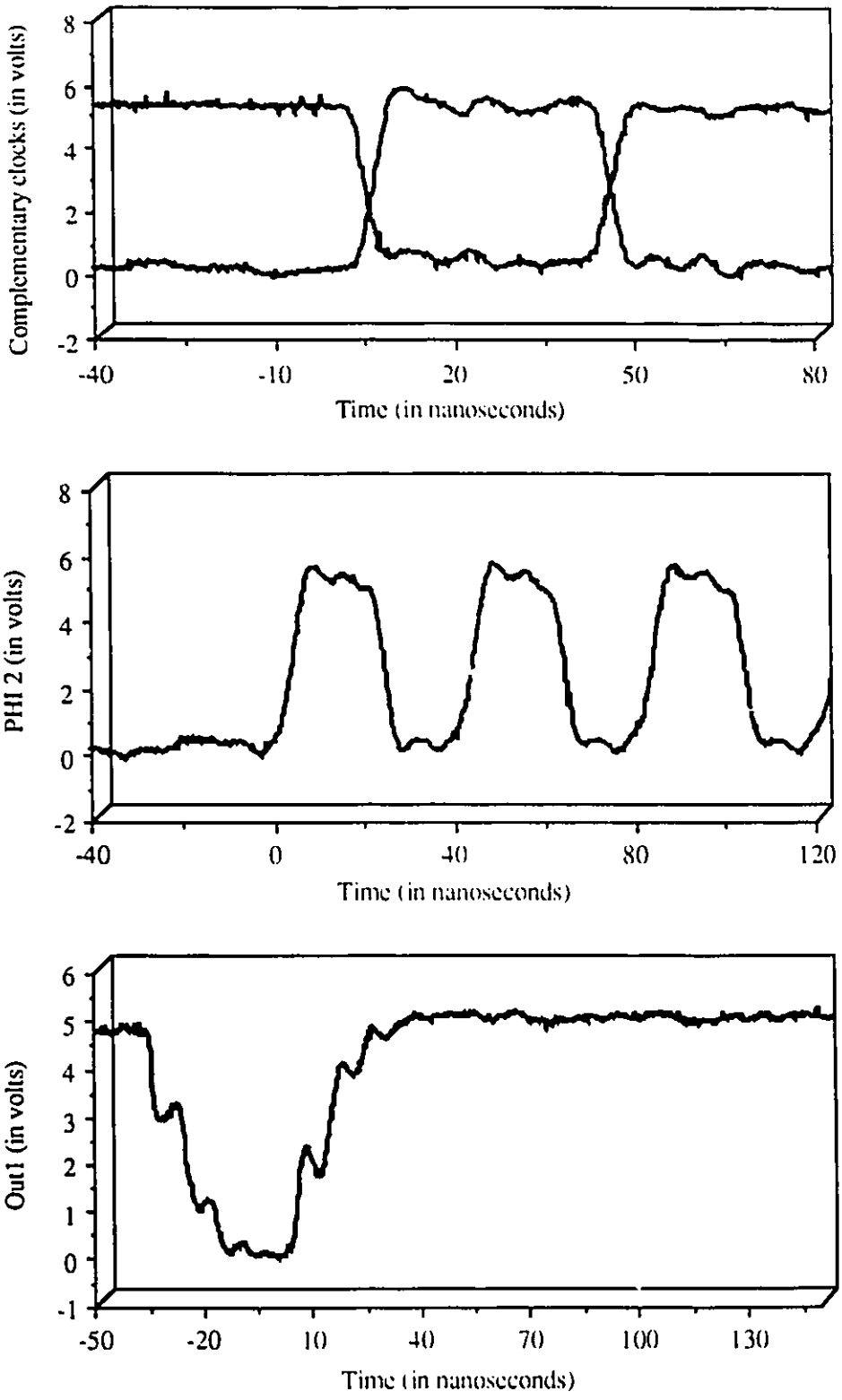


Fig F.10: PSR waveforms.

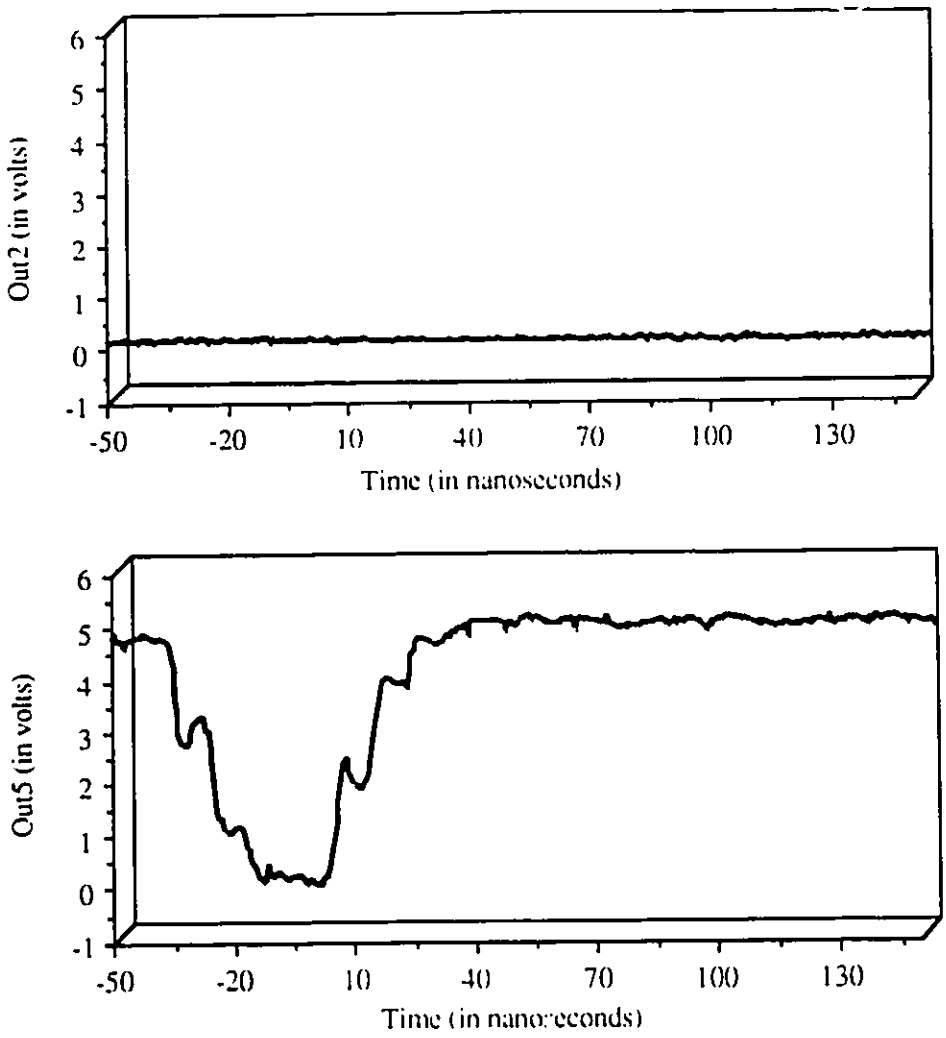


Fig F.10: Continued.

Vita Auctoris

Jamal Benzreba was born on September 1, 1964 in Tripoli, Libya. He completed his high school education at Sook El Jomma, Tripoli in 1982. He then obtained an ESL certificate from Sudbury, Ontario in 1985. He graduated from McMaster university in 1989 with a Bachelor of Engineering in Engineering Physics. In September 1991 he finished his Masters of Applied Science in Electrical Engineering at the University of Windsor. His interests include soccer, chess, and table tennis.